**Ph.D. DISSERTATION**

# A study on algorithms for learning spiking neural network in neuromorphic hardware

**by**

**Dohun KIM**

**August 2021**

**DEPARTMENT OF MATERIALS SCIENCE AND ENGINEERING**
**COLLEGE OF ENGINEERING**
**SEOUL NATIONAL UNIVERSITY**

# A study on algorithms for learning spiking neural network in neuromorphic hardware

Advisor : Prof. Cheol Seong Hwang

by

DOHUN KIM

A thesis submitted to the Graduate Faculty of Seoul National University in partial fulfillment of the requirements for the
Degree of Doctor of Philosophy
Department of Materials Science and Engineering

August 2021

Approved
by

Chairman of Advisory Committee : Sangbum Kim
Vice-chairman of Advisory Committee : Cheol Seong Hwang
Advisory Committee : Dongsuk Jeon
Advisory Committee : Doo Seok Jeong
Advisory Committee : Jaewook Kim

# Abstract

Spiking neural networks (SNNs) are believed to offer solutions to biologically inspired and energy-efficient computation. SNNs are dynamics models that process and convey data by means of asynchronous spike events. The spikes are sparse in time and space and have high information content. The rich dynamics of SNNs enable the effective learning of complex spatiotemporal firing patterns in a dynamic domain. However, internal state updates, e.g., membrane potential, are required every timestep, which requires a lot of computation time. Thus, the internal state updates of neurons must be processed in parallel for efficient spiking simulation. Distributed processors and local memories enable this parallel computation in dedicated neuromorphic hardware. Event-based weight update using local data can maximize the computational efficiency of neuromorphic hardware. However, the universal SNN learning algorithm based on the event and local data is still missing, especially associative recall.

In this paper, we introduce an *n*th order sequence-predicting SNN (*n*-SPSNN), which is capable of single-step prediction and sequence-to-sequence prediction, i.e., associative recall. As a key to these capabilities, we propose a new learning algorithm, named the learning by backpropagating action potential (LbAP) algorithm, which features (i) postsynaptic event-driven learning, (ii) access to topological and temporal local data only, (iii) competition-induced weight normalization effect, and (iv) fast learning. Most importantly, the LbAP algorithm offers a unified learning framework over the entire SPSNN based on local data only. The learning capacity of the SPSNN is mainly dictated by the number of hidden neurons *h*; its prediction accuracy reaches its maximum value (~1) when the hidden neuron number *h* is larger than twice training sequence length *l*, i.e., $h \geq 2l$. Another advantage is its high tolerance to errors in input encoding compared to the state-of-the-art sequence learning networks, namely long short-term memory (LSTM) and gated recurrent unit (GRU). Additionally, its efficiency in learning is approximately 100 times that of LSTM and GRU when measured in terms of the number of synaptic operations until successful training, which corresponds to multiply-accumulate

operations for LSTM and GRU. This high efficiency arises from the higher learning rate of the SPSNN, which is attributed to the LbAP algorithm.

Applying a nonvolatile memory to neuromorphic hardware leverage the computational efficiency in matrix-vector multiplication. Resistance switch is a promising candidate for nonvolatile memory. The binary resistance switch array implements efficient matrix-vector multiplication by measuring the output current vector to the applied input voltage. The spike propagation in SNNs can be applied to the matrix-vector multiplication in the resistive switch array. Thus, the parallel computation can be accelerated when implementing an artificial synapse array with a binary resistance switch array.

However, SNNs require synaptic weights with multi-bit precision, which is not suitable for neuromorphic hardware using binary resistance switches. Also, using multi-bit precision on neuromorphic hardware increases the memory footprint and reduces computational efficiency. In this regard, we propose a novel event-based weight binarization (eWB) algorithm for SNNs with binary synaptic weights (-1, 1). The eWB algorithm is based on the Lagrange multiplier method, which optimizes parameters within given constraints. The algorithm features (i) event-based asymptotic weight binarization using local data only, (ii) full compatibility with event-based learning algorithms (e.g., spike timing-dependent plasticity and event-driven random backpropagation (eRBP) algorithm), and (iii) the capability to address various constraints (including the binary weight constraint). As a proof of concept, we combine eWB with eRBP (eWB-eRBP) to obtain a single algorithm for learning binary weights to generate correct classifications. Fully connected SNNs were trained using eWB-eRBP and achieved an accuracy of 95.35% on MNIST.

---

**Keywords:** neuromorphic engineering, resistance switch array, spiking neural networks, event-driven learning algorithm of locality, sequence learning, associative recall, event-based weight binarization

**Student Number:** 2016-20771

Dohun KIM

# Table of Contents

# List of Tables

# List of Figures

# List of Abbreviations

| | |
|---|---|
| bAPs | Backpropagating action potentials |
| BP | Backpropagation |
| BNN | Binarized neural network |
| BWN | Binary weight network |
| CD | Contrastive divergence |
| CFS | Constraint failure score |
| DNN | Deep neural network |
| eCD | Event-driven CD |
| eRBP | Event-driven random backpropagation |
| eWB | Event-based weight binarization |
| fWB | Forced-to-be-binary weights |
| GRU | Gated recurrent unit |
| HB-STDP | Hybrid-STDP |
| HTM | Hierarchical temporal memory |
| IF | Integrate-and-fire |
| LbAP | Learning by backpropagating action potential |
| LIF | Leaky integrate-and-fire |
| LSTM | Long short-term memory |
| LMM | Lagrange multiplier method |
| LTD | Long-term depression |
| LTP | Long-term potentiation |
| MAC | Multiply-accumulate |
| *n*-SPSNN | *n*th order sequence-predicting SNN |
| SNN | Spiking neural network |
| SRM | Spike-response model |

| | |
|---|---|
| STDP | Spike timing-dependent plasticity |
| SynOps | Synaptic operations |
| RNN | Recurrent neural network |

# 1. Introduction

## 1.1. Spiking neural networks (SNNs)

The human brain is a complex system with approximately 100 billion neurons and trillions of interconnected synapses [1]. Neuronal information is conveyed by asynchronous spikes without forward locking. Spike-based temporal sparse processing enables efficient information transfer in the brain.

Deep neural networks (DNNs) (Fig 1.1(a)) aim to mimic the behavior of a biological nervous system. DNNs are typically trained by error-backpropagation algorithms (BP) for layer-wise weight updates. DNNs are very powerful learning models that solve complex problems such as visual image recognition [2-4], speech recognition [5] and controlling tasks [6] beyond the human level. Despite the ongoing success, the substantial computational cost compared to the human brain and their inability to capture the temporal correlation of neural activities have created a need for more biologically plausible learning algorithms.

Spiking neural networks (SNNs) are the primary candidate for realizing neuromorphic systems that require lower computational effort based on temporal coding. SNNs are dynamic models in which neuronal information is processed in form of spikes, as shown in Fig 1.1(b). Spike is an essentially binary event, either 0 or 1. A spiking neuron in SNNs is only active when it receives or emits spikes. This unique sparse event-driven processing using neuronal computation and synaptic weight updates improves energy efficiency in neuromorphic hardware implementations [7, 8].

The rich spatio-temporal dynamics of SNNs stem from temporal kernels. The time course of internal state in spiking neuron models results from convolutions of time-varying input stimulus, as shown in the spike-response model (SRM) [9]. The rich dynamics of SNNs enable the effective learning of complex spatio-temporal spiking patterns in a dynamic domain. The previously proposed remote supervised method (ReSuMe) [10], chronotron [11], spike pattern association neuron (SPAN) [12], and precise-spike-driven synaptic plasticity (PSD) [13] rules have both demonstrated

success in learning SNN to predict precise temporal representations of spatiotemporal spike patterns. However, success in sequence-to-sequence prediction using these learning rules has not been demonstrated.

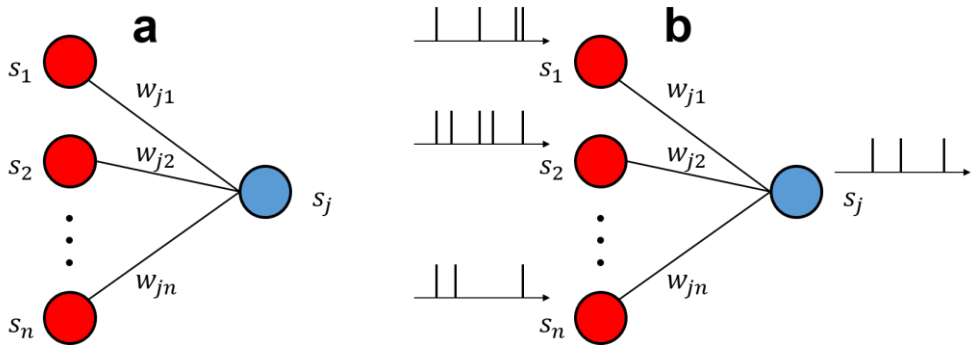


Figure 1.1. Illustration of (a) an ANN and (b) an SNN. In SNN, spiking neurons communicate binary sparse spikes in the temporal domain.

## 1.2. Dedicated hardware for spiking neural network

Neuromorphic Engineering originally aimed to implement biologically plausible SNNs using very large integrated analog circuits [14]. SpiNNaker [15], Neurogrid [16], TrueNorth [17], DYNAPs [18], and Loihi [19] are recently released prototypes of neuromorphic hardware. Although their working principles and capabilities are different, they realize SNNs on the chips. TrueNorth, SpiNNaker, and Loihi are based on digital circuits that allow flexibility of network configuration and learning algorithms.

The neuromorphic hardware has shown potential in accelerating SNN simulations with the reconfigurable network topology and learning algorithms. Fig. 1.2 shows the ideal architecture of digital neuromorphic hardware for SNNs. The memory is distributed over neurons as opposed to the von Neumann architecture in which processing units and memory are separated. Each processing unit calculates the state variables such as synaptic current and membrane potential. The memory for each processing unit contains its local data. The information of spiking events is processed across the cores through data buses. The parallel computation realized by distributed processors and local memories maximizes synaptic operation speed, the key to neuromorphic hardware performance. Thus, digital neuromorphic hardware is deemed to leverage the capacity of an event-driven local algorithm for SNNs.

Representing synaptic weights in digital neuromorphic hardware using non-volatile memory can provide higher connectivity and faster computation speed. Resistance switch is considered as one of the most promising candidates for next-generation non-volatile memory [20]. The resistance switch device has a high resistance state (HRS) and low resistance state (LRS). The resistance state can be reversibly switched between HRS and LRS under sufficient voltage stimuli. Binary resistance switch array (Fig. 1.3) has demonstrated an efficient hardware implementation of the matrix-vector multiplication by measuring the output current to an applied input voltage vector [21]. Also, the non-volatile characteristic of the resistance switch allows energy efficiency compared to the volatile memories, e.g., dynamic random-access memory (DRAM). Thus, it is possible to implement parallel

matrix operations and reduce energy consumption when the binary resistance switch array is applied to digital neuromorphic hardware.

In this regard, this paper consists of two parts. At first, we introduce a novel learning algorithm for SNNs and an SNN architecture for sequence prediction, which are suitable for neuromorphic hardware. In the second part, we propose a new learning algorithm called the event-based weight binarization (eWB) algorithm, which can be implemented in digital neuromorphic hardware with binary resistance switch array.

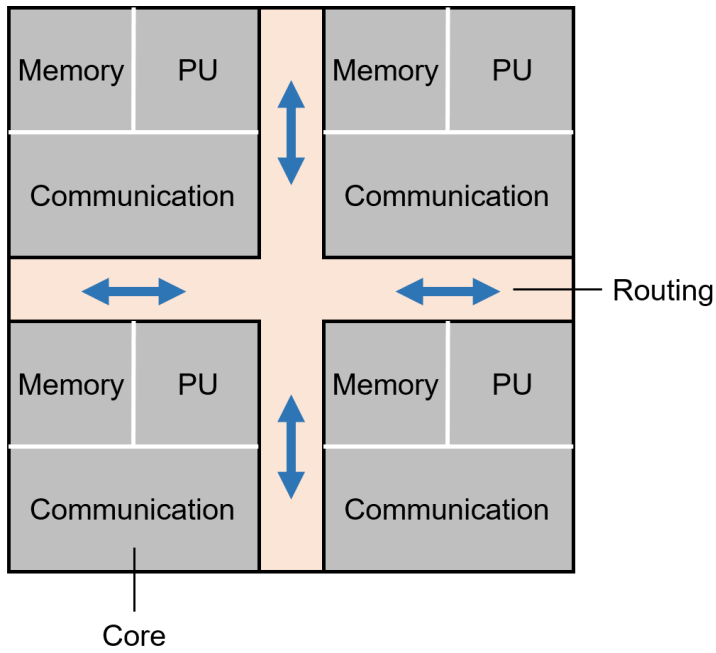


Figure 1.2. Schematic of the ideal architecture of digital neuromorphic cores. PU denotes a procession unit.

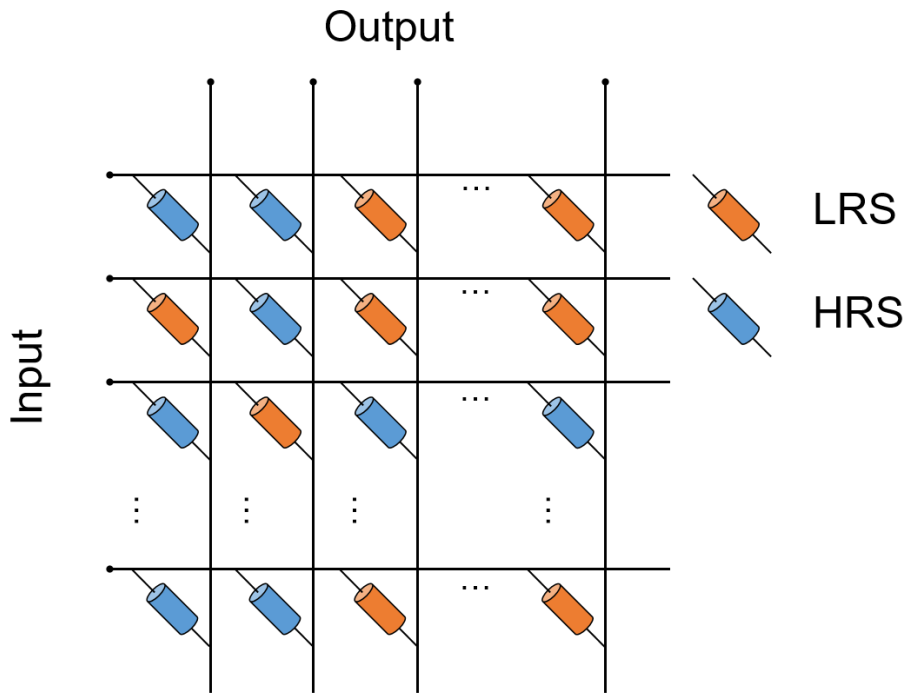


Figure 1.3. Schematic of resistance switch array.

## 1.3. Bibliography


[1] F. A. Azevedo *et al.*, *Journal of Comparative Neurology,* vol. 513, no. 5, pp. 532-541, 2009.

[2] D. Cireşan, U. Meier, and J. Schmidhuber, *arXiv preprint arXiv:1202.2745,* 2012.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Advances in neural information processing systems*, 2012, pp. 1097-1105.

[4] Y. Taigman, M. Yang, M. A. Ranzato, and L. Wolf, *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1701-1708.

[5] G. E. Dahl, D. Yu, L. Deng, and A. Acero, *IEEE Transactions on audio, speech, and language processing,* vol. 20, no. 1, pp. 30-42, 2011.

[6] D. Silver *et al.*, *nature,* vol. 529, no. 7587, p. 484, 2016.

[7] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, *Frontiers in neuroscience,* vol. 13, p. 95, 2019.

[8] C. Lee, S. S. Sarwar, and K. Roy, *arXiv preprint arXiv:1903.06379,* 2019.

[9] W. Gerstner and W. M. Kistler. Cambridge University Press, 2002.

[10] F. Ponulak and A. Kasiński, *Neural Comput.,* vol. 22, no. 2, pp. 467-510, 2010.

[11] R. V. Florian, *PloS one,* vol. 7, no. 8, 2012.

[12] A. Mohemmed, S. Schliebs, S. Matsuda, and N. Kasabov, *International journal of neural systems,* vol. 22, no. 04, p. 1250012, 2012.

[13] Q. Yu, H. Tang, K. C. Tan, and H. Li, *Plos one,* vol. 8, no. 11, 2013.

[14] C. Mead, *Proceedings of the IEEE,* vol. 78, no. 10, pp. 1629-1636, 1990.

[15] E. Painkras *et al.*, *IEEE Journal of Solid-State Circuits,* vol. 48, no. 8, pp. 1943-1953, 2013.

[16] B. V. Benjamin *et al.*, *Proceedings of the IEEE,* vol. 102, no. 5, pp. 699-716, 2014.

[17] P. A. Merolla *et al.*, *Science,* vol. 345, no. 6197, pp. 668-673, 2014.

[18] N. Q. S. Moradi, F. Stefanini, G. Indiveri, *IEEE Trans. Biomed. Circuits Syst.,* vol. 12, pp. 106-122, 2018.

[19] M. Davies *et al.*, *IEEE Micro,* vol. 38, no. 1, pp. 82-99, 2018.

[20] A. Beck, J. Bednorz, C. Gerber, C. Rossel, and D. Widmer, *Applied Physics Letters,* vol. 77, no. 1, pp. 139-141, 2000.

[21] M. Hu, H. Li, Q. Wu, and G. S. Rose, *DAC Design Automation Conference 2012*, 2012: IEEE, pp. 498-503.

# 2. Literature

## 2.1. Sequence-predicting SNN

Associative memory or recall has long been the subject of research interest in sequence learning, which is categorized as sequence-to-sequence learning. The Hopfield network is a seminal network to this end; the recurrent network is given the capability to memorize patterns over the neurons, and a piece of a pattern can activate the whole pattern [1]. However, the network still lacks dynamics, and hence, the sequence information of the pattern is ignored. By contrast, RNNs are dynamic hypotheses that can learn sequences and make single-step predictions. This capability is given by the feedback connection that bases the current prediction on the network activity at the previous time step [2]. However, training the network through time causes several critical issues such as vanishing gradient and exploding gradient problems [3]. Variations of RNN, e.g., LSTM [4] and GRU [5], cope with these issues due to a constant error-flow through internal units. A backpropagation algorithm is commonly used to train these networks. The good performance of these networks comes at the cost of large computational power [6]. In particular, their demands for computational power are highlighted by comparison with the *n*-SPSNN, which will be addressed in Section 3.3.4. Their low tolerance to errors in sequence encoding is another disadvantage.

The sequence-learning neural network proposed by Wang and Yuwono [7] employs short-term memory networks as sub-networks, which play a similar role to the working memory of the *n*-SPSNN in sequence learning. The short-term memory network considers memory decay with time so that a discount factor applies to the contributions of previous elements to the present element prediction. Each subsequence of previous elements is mapped onto a single detector in an injective manner, given strong lateral inhibition among the detectors, which is also similar to the proposed *n*-SPSNN architecture. However, the neural network proposed by Wang and Yuwono consists of binary neurons, i.e., McCulloch–Pitts neurons, and the inhomogeneous learning rule applying to the network fails to provide a unified framework of learning. The learning rule needs to access global data (all weights

values) to normalize the weight under update, which differs from our LbAP algorithm. Unfortunately, the sequence-learning capacity of the network is unavailable.

SNNs with recurrent connections are considered as hypotheses for associative recall [8-10]. Unlike the associative recall in the above-mentioned class, the associative recall in SNNs concerns not only a sequence of spiking neurons but also their precise spike times. This is because SNNs are endowed with the capacity to learn dynamic-domain data given their rich dynamics. Pfister et al. considered an SNN with stochastic spiking neurons and proposed a method to train the SNN to lead the postsynaptic neurons to fire spikes at desired times [11]. This work was further extended to more complicated SNNs with visible and hidden neurons, where the hidden neurons receive supervision signals and represent a sequence [8]. In this work, the objective function for weight optimization was defined as the difference in a distribution function between a desired and actual spike time. The proposed learning algorithm for the visible neurons is found to be equivalent to the voltage-based STDP rule [12] in support of the physiological fidelity of the proposed learning algorithm. However, a different learning algorithm is used for the hidden neurons, which fails to provide a unified learning framework. Unfortunately, systematic analyses on the performance (learning capacity, efficiency, error-tolerance, etc.) of the recurrent SNN are unavailable.

Gardner and Grüning modified the learning rule by Pfister et al. [11] to train an SNN of deterministic neurons, referred to as FILT [13]. In the FILT rule, the synaptic weight is adjusted to reduce the difference in spike filtering between a desired and actual spike train. In a similar framework, several learning algorithms have been proposed to produce a desired spike train, such as ReSuMe [9], chronotron [14], SPAN [15], and PSD synaptic plasticity [16]. These learning rules are capable of training a neuron to generate the desired spike train in response to the input spike pattern. However, success in associative recall using these learning rules has not been demonstrated.

An STDP rule in conjunction with heterosynaptic depression enables a recurrent SNN to form synaptic chains, each of which represents a sequence that is recalled associatively [10]. Such chains are formed at random; however, the number of chains tends to decrease with the strength of global inhibition. In addition, the network can copy an applied sequential input during training and reproduce the input subsequently. However, a critical downside is the necessary access to global data, such as total synaptic weights (for the heterosynaptic depression) and activity state variables (for global inhibition), which is inconsistent with the attributes of an ideal learning rule embedded in neuromorphic hardware.

The hierarchical temporal memory (HTM) adopts and simplifies the physiological observation that, in a pyramidal cell, a delay in postsynaptic potential is proportional to the distance between the dendritic spine and soma [17, 18]. The HTM network consists of a set of columns; a combination of such columns represents an element in a sequence. A sequence is learned such that a synaptic chain (with the same length as the sequence) representing the sequence is formed in the network. Therefore, learning complex sequences is inefficient.

## 2.2. Binarized SNN

To date, the proposed methods for learning binary weights in SNNs can be mainly classified into two groups: (i) conversion of binarized neural networks (BNNs) [19] or binary weight networks (BWNs) [20] into SNNs and (ii) probabilistic learning. The first approach is motivated by recent successes in weight binarization in DNNs [19-22]. Lu and Sengupta [23] proposed a method to map BWNs onto VGG-15-like SNNs with perfect integrate-and-fire (IF) neurons. The IF neuron is considered to encode an input spike train as a firing rate, thereby ensuring its similarity to a rectified linear unit (ReLU).

The second approach uses weights that probabilistically toggle between the binary weights 0 and 1. However, it is common to use auxiliary variables to determine the probability of weight updates. Suri et al. [24] proposed an STDP-based stochastic algorithm using binary weights. Potentiation (0→1) and depression (1→0) occur with probabilities based on the temporal order of presynaptic and postsynaptic spikes.

Nevertheless, a challenge in STDP is its inability to scale to deep SNNs. STDP is hence commonly limited to shallow SNNs. Yousefzadeh et al. [25] also proposed an STDP-based stochastic algorithm to learn binary weights. The algorithm requires additional operations such as weight normalization and threshold adjustment for individual neurons. The considered network is shallow (one feature extraction layer and one classifier) because of the aforementioned limitation of STDP. Additionally, it does not support end-to-end training. Srinivasan and Roy [26] proposed the Hybrid-STDP (HB-STDP) algorithm based on probabilistic STDP. Notably, HB-STDP includes a dead zone in the STDP time window, in which neither potentiation nor depression is allowed. HB-STDP captures temporally correlated inputs by preventing excessive potentiation and depression; however, its low accuracy is a challenge.

Nevertheless, a challenge in STDP is its inability to scale to deep SNNs. STDP is hence commonly limited to shallow SNNs. Yousefzadeh et al. [25] also proposed an STDP-based stochastic algorithm to learn binary weights. The algorithm requires

additional operations such as weight normalization and threshold adjustment for individual neurons. The considered network is shallow (one feature extraction layer and one classifier) because of the aforementioned limitation of STDP. Additionally, it does not support end-to-end training. Srinivasan and Roy [26] proposed the Hybrid-STDP (HB-STDP) algorithm based on probabilistic STDP. Notably, HB-STDP includes a dead zone in the STDP time window, in which neither potentiation nor depression is allowed. HB-STDP captures temporally correlated inputs by preventing excessive potentiation and depression; however, its low accuracy is a challenge.

The learning of low-precision weights in SNNs as generative models is a subject of interest. Stromatias et al. [27] tailored contrastive divergence (CD) to deep belief networks with spiking neurons. The original double-precision floating-point weights are converted to a lower precision floating-point format to reduce memory consumption. However, the reduction in precision comes with the cost of significant performance degradation. Neftci et al. [28] proposed the event-driven CD (eCD) algorithm to train restricted Boltzmann machines with spiking neurons. In eCD, the weight update is fine-tuned by STDP. Low-precision (down to 2-bit) weights were tested; however, a significant reduction in accuracy was unavoidable.

The eWB algorithm adopts an approach that parameterizes the degree of binary-constraint fulfillment and asymptotically optimizes the degree upon the occurrence of events, whereas the aforementioned precision-reduction methods merely round the full-precision weights. Therefore, eWB cannot be classified as any of the aforementioned approaches.

## 2.3. Bibliography


[1] J. J. Hopfield, *Proc. Natl. Acad. Sci.,* vol. 79, no. 8, pp. 2554-2558, 1982.

[2] P. J. Angeline, G. M. Saunders, and J. B. Pollack, *IEEE transactions on Neural Networks,* vol. 5, no. 1, pp. 54-65, 1994.

[3] Y. Bengio, P. Simard, and P. Frasconi, *IEEE transactions on neural networks,* vol. 5, no. 2, pp. 157-166, 1994.

[4] S. Hochreiter and J. Schmidhuber, *Neural computation,* vol. 9, no. 8, pp. 1735-1780, 1997.

[5] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, *arXiv preprint arXiv:1412.3555,* 2014.

[6] J. S.-D. Jasmine Collins, David Sussillo, *34th International Conference on Machine Learning*, Sydney, Australia, 2017.

[7] D. L. Wang and B. Yuwono, *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 25, no. 4, pp. 615-628, 1995.

[8] J. Brea, W. Senn, and J.-P. Pfister, *J. Neurosci.,* vol. 33, no. 23, pp. 9565-9575, 2013.

[9] F. Ponulak and A. Kasiński, *Neural Comput.,* vol. 22, no. 2, pp. 467-510, 2010.

[10] I. R. Fiete, W. Senn, C. Z. H. Wang, and R. H. R. Hahnloser, *Neuron,* vol. 65, no. 4, pp. 563-576, 2010.

[11] J.-P. Pfister, T. Toyoizumi, D. Barber, and W. Gerstner, *Neural computation,* vol. 18, no. 6, pp. 1318-1348, 2006.

[12] C. Clopath, L. Büsing, E. Vasilaki, and W. Gerstner, *Nat. Neurosci.,* Article vol. 13, p. 344, 2010.

[13] B. Gardner and A. Grüning, *PloS one,* vol. 11, no. 8, p. e0161335, 2016.

[14] R. V. Florian, *PloS one,* vol. 7, no. 8, 2012.

[15] A. Mohemmed, S. Schliebs, S. Matsuda, and N. Kasabov, *International journal of neural systems,* vol. 22, no. 04, p. 1250012, 2012.

[16] Q. Yu, H. Tang, K. C. Tan, and H. Li, *Plos one,* vol. 8, no. 11, 2013.

[17] J. Hawkins and S. Ahmad, *Front. Neural Circuits,* Hypothesis & Theory vol. 10, no. 23, 2016.

[18] Y. Cui, S. Ahmad, and J. Hawkins, *Neural Comput.,* vol. 28, no. 11, pp. 2474-2504, 2016.

[19] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, 2016, *arXiv:1602.02830*.

[20] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, *European conference on computer vision*, 2016: Springer, pp. 525-542.

[21] M. Courbariaux, Y. Bengio, and J.-P. David, *Advances in neural information processing systems*, 2015, pp. 3123-3131.

[22] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, *The Journal of Machine Learning Research,* vol. 18, no. 1, pp. 6869-6898, 2017.

[23] S. Lu and A. Sengupta, 2020, *arXiv:2002.10064*.

[24] M. Suri *et al.*, *IEEE Transactions on Electron Devices,* vol. 60, no. 7, pp. 2402-2409, 2013.

[25] A. Yousefzadeh, E. Stromatias, M. Soto, T. Serrano-Gotarredona, and B. Linares-Barranco, *Frontiers in neuroscience,* vol. 12, p. 665, 2018.

[26] G. Srinivasan and K. Roy, *Frontiers in Neuroscience,* vol. 13, p. 189, 2019.

[27] E. Stromatias, D. Neil, M. Pfeiffer, F. Galluppi, S. B. Furber, and S.-C. Liu,

*Frontiers in neuroscience,* vol. 9, p. 222, 2015.

[28] E. O. Neftci, B. U. Pedroni, S. Joshi, M. Al-Shedivat, and G. Cauwenberghs, *Frontiers in neuroscience,* vol. 10, p. 241, 2016.

# 3. SPSNN: *n*th order sequence-predicting spiking neural network

## 3.1. Introduction

Spiking neural network (SNN) is a dynamic hypothesis with diverse temporal kernels to express neuronal behaviors in response to synaptic transmission [1-3]. The central nervous system (CNS) is based on the SNN, and the SNN has therefore been analyzed theoretically to understand the working principles of the CNS. Apart from the SNN's physiological plausibility, its feasible applications to deep learning as a hypothesis have attracted considerable research attention from various fields [4-9]. The effort to realize an SNN using integrated circuits—which has continued over the last three decades—paves the way for the data- and energy-efficient acceleration of deep learning. This has been emerging as an important goal of neuromorphic engineering [5, 6]. In this case, the main challenge lies in the learning algorithm; a universal learning algorithm, similar to backpropagation algorithms for deep neural network (DNN), is still missing. There exist several methods to optimize synaptic weights in an SNN. They usually map the weights of backpropagation-trained DNNs onto SNNs [10, 11]. However, to leverage the SNN in neuromorphic hardware, the learning needs to be based on an event-driven algorithm of locality [5, 6, 12]. For instance, the event-driven random backpropagation (eRBP) algorithm satisfies this requirement considering that (i) the ad hoc update is driven by a presynaptic spike and (ii) only local variables are used to evaluate the change in weight [13]. Static-domain data, e.g., images, are suited to the eRBP algorithm for training SNNs.

Considering the rich dynamics of SNN, learning with dynamic-domain data perhaps harnesses the full capability of SNNs [3, 12]. Dynamic-domain data include time-series data, which embody periodic discrete data points in a time domain. In a framework of deep learning, the recurrent neural network (RNN) and its variations, e.g., long short-term memory (LSTM) [14] and gated recurrent unit (GRU) [15], are known to have an excellent capability to learn time-series data. Unfortunately, there is a lack of both SNN architecture for learning time-series data as well as a learning

algorithm for the architecture, which performs sequence-prediction tasks, e.g., single-step prediction and sequence-to-sequence prediction (also known as associative recall), with accuracy comparable to that of LSTM and GRU.

In this regard, we propose an SNN architecture for temporal sequence learning, named *n*th order sequence-predicting spiking neural network (*n*-SPSNN). The indispensable working memory for the prediction is realized using synaptic chains. To train the *n*-SPSNN, we propose an event-driven learning algorithm of locality, referred to as learning by backpropagation action potential (LbAP) algorithm. The LbAP algorithm was inspired by physiological observations of backpropagating action potential (bAP) boosts intervening in homosynaptic plasticity [16]. Note that the weight is only updated upon postsynaptic events in contrast to other event-driven algorithms such as spike timing-dependent plasticity (STDP) rule (updates upon both presynaptic and postsynaptic events) and eRBP algorithm (updates upon presynaptic events only). The locality allows the LbAP algorithm to be suitable for memory-efficient implementation in digital neuromorphic hardware, particularly, multi-core neuromorphic processors [6, 12, 17, 18].

.

## 3.2. Sequence-predicting spiking neural network and learning algorithm

### 3.2.1. Sequecne prediction principle and network architecture

We define a sequence of $l$ elements as ($x_1, x_2, \ldots, x_l$), where each element is chosen from a set of $m$ symbols $S$ (=$\{s_1, s_2, \ldots, s_m\}$), i.e., $x_i \in \{s_1, s_2, \ldots, s_m\}$. $\mathrm{x_i} \in \{\mathrm{s_1}, \mathrm{s_2}, \ldots, \mathrm{s_m}\}\mathrm{x_i} \in \{\mathrm{s_1}, \mathrm{s_2}, \ldots, \mathrm{s_m}\}x_i$ and $x_{i+1}$ are separated by $\Delta t_e$ in time. Each element is represented by an $m$-long one-hot vector. The $n$-SPSNN learning a sequence is illustrated in Fig. 3.1(a). The network is given $m$ parallel sub-networks, each with a synaptic chain of $n$ neurons. One sub-network is dedicated to one of $m$ symbols only. The parallel $m$ sub-networks are in full connection with a hidden layer loaded with $h$ neurons. A weight matrix for the feedforward connections is defined as $\boldsymbol{w}_1$ ($\boldsymbol{w_1} \in \mathbb{R}^{h \times nm}$; $w_1[i,j] \in [0, w_{max1}]$). The hidden neurons are fully connected with $m$ output neurons in an output layer. Each output neuron represents each of $m$ elements. Weight matrix $\boldsymbol{w}_2$ ( $\boldsymbol{w_2} \in \mathbb{R}^{m \times h}$; $w_2[i,j] \in [0, w_{max2}]$ ) defines the feedforward connections. Note that full lateral inhibition applies to both hidden and output layers. This $n$-SPSNN network is expressed as $m$-($n$×$m$)-$h$-$m$, considering the number of neurons in each layer

The element on a given time step in a sequence is encoded as a one-hot vector. Each element of the vector is subsequently applied to the input neuron of the corresponding sub-network such that "1" indicates the presence of an input spike, whereas "0" indicates no spike (see Fig. 3.1(a)). Each spike is relayed over the synaptic chain in each sub-network. We consider an axonal delay ($\Delta t_{ax1}$) between neighboring neurons in a synaptic chain of $n$ neurons. Assuming that $\Delta t_{ax1} = \Delta t_e$, a spike on a given time step hops to the next neuron in $\Delta t_e$, and simultaneously, the next element in the sequence arrives at the input neurons. Therefore, a spike representing a particular element on a given time step can stay in the sub-network over the synaptic chain for ($n$-1) $\Delta t_{ax1}$, serving as a working memory. Unless otherwise stated, the equalities $\Delta t_{ax1} = \Delta t_e$ and $\Delta t_e$ = 100 ms hold. Note that the $n$-SPSNN robustly predicts a sequence with random variations in $\Delta t_e$ (i.e., $\Delta t_{ax1} \neq \Delta t_e$), which will be addressed in Section 3.3.3.

Fig. 3.1(b) schematizes this process for a 4-SPSNN ($n$ = 4) trained with an arbitrary sequence of (B, A, C, D, B, B, C) ($l$ = 7; $m$ = 4; $S$ = {A, B, C, D}). Sub-nets 1, 2, 3, and 4 represent elements A, B, C, and D, respectively. The table for each sub-network in Fig. 3.1(b) indicates a neuronal activity vector on each time step such that a non-zero element and "0" denote the presence of a spike and no spike, respectively. For instance, $000A_1$ for Sub-net 1 at $t_2$ means that Neuron 1 (N1) fires a spike, while the rest of the neurons (N2, N3, and N4) are quiet. Here, $A_1$ indicates the element corresponding to the sub-network (A) and the neuron index (subscript). The $n$-SPSNN begins sequence prediction when $n$ preceding elements are available. Thus, the 4-SPSNN in Fig. 3.1(b) begins the prediction at $t_4$ based on the component-wise sum of the four neuronal activity vectors ($B_4A_3C_2D_1$), meaning that N3 (Sub-net 1), N4 (Sub-net 2), N2 (Sub-net 3), and N1 (Sub-net 4) fire spikes at $t_4$ simultaneously. The prediction at $t_5$ is based on the vector ($A_4C_3D_2B_1$), meaning the simultaneous spiking of N4 (Sub-net 1), N1 (Sub-net 2), N3 (Sub-net 3), and N2 (Sub-net 4).

However, early prediction before seeing $n$ previous elements is made occasionally in real network operations, as will be addressed in Section 3.3.1.

Based on an $n$-long neuronal activity vector at $t_i$, the $n$-SPSNN should be able to predict the input element at $t_{i+1}$. For instance, $B_4A_3C_2D_1$ at $t_4$ in Fig. 3.1(b) outputs B as a predicted element at $t_5$ such that the output neuronal activity vector is 0100. The hidden layer in full connection with the parallel sub-networks and output layer associates the $n$-long neuronal activity vector with the desired output neuronal activity vector. Hidden neurons need to detect $n$ simultaneous spikes on a given time step and fire a spike accordingly, serving as a coincident detector. The spiking pattern of hidden neurons should be specific to a certain spatial pattern of spiking over the sub-networks. For instance, $B_4A_3C_2D_1$ at $t_4$ and $A_4C_3D_2B_1$ at $t_5$ should cause different spiking patterns to distinguish them. The lateral inhibition over the hidden neurons suppresses overlap between different spiking patterns. Eventually, the spiking pattern of hidden neurons activates the desired output neuron through the feedforward connections. The lateral inhibition over the output neurons ensures the clear separation of a desired output neuron from the others.

For associative recall (sequence-to-sequence prediction), the *n*-SPSNN for single-step prediction is modified to employ feedback connection from the output to the input layer. The single-step prediction in response to a set of *n* preceding elements is fed into the input layer as an input. This feedback process continues onward until the end of the sequence.

The hidden and output neurons are expressed as a multi-compartment model in that the dendritic and somatic potentials are separately evaluated for each neuron. Both potentials are evaluated using a SRM [2] (Appendix 3.5). However, the dendrite is not allowed to fire spikes so that no refractory kernel applies to the dendritic potential evaluation. We consider axonal delays for the sub-networks-to-hidden layer and hidden layer-to-output layer feedforward connections, which are $\Delta t_{ax2}$ and $\Delta t_{ax3}$, respectively. They are fixed to 20 ms.

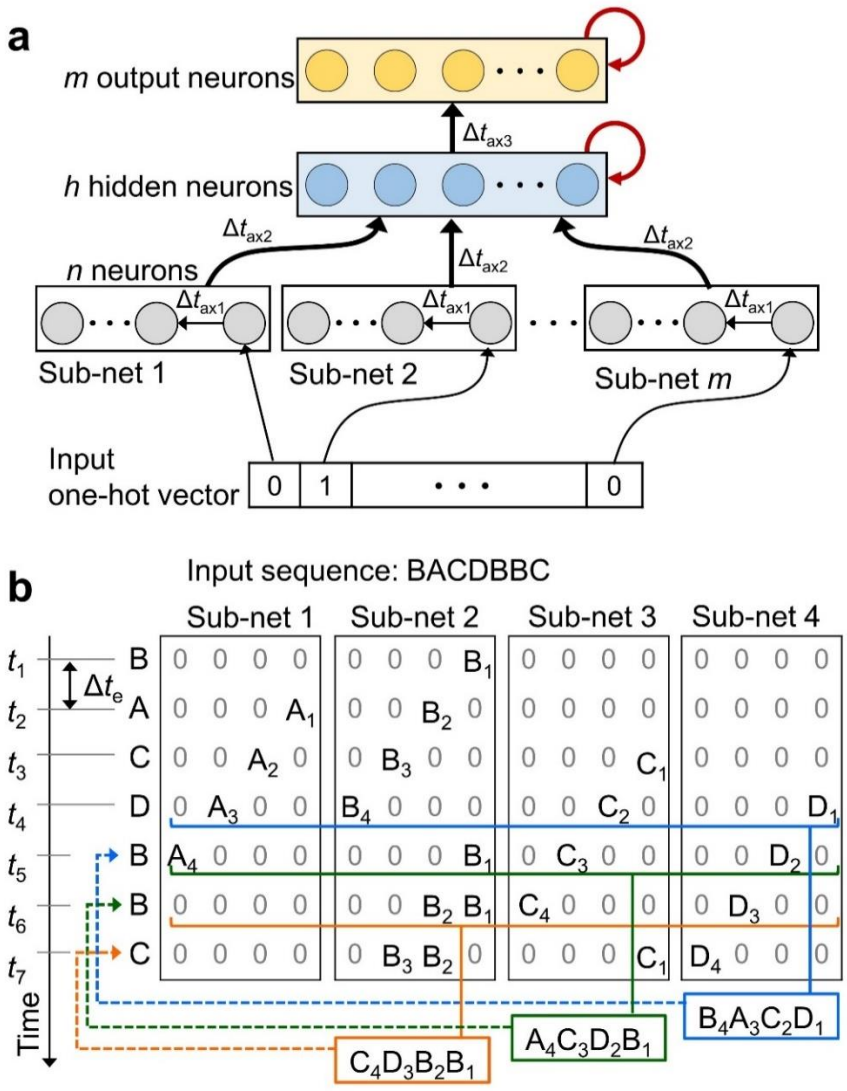


Figure 3.1. (a) Schematic of the *n*-SPSN (*m*-(*n*×*m*)-*h*-*m*). The thick arrows indicate all-to-all connection, whereas the thin arrows indicate element-to-element connections. Lateral inhibition is indicated by red arrows. (b) Visualized single-step predictions for a sequence of (B, A, C, D, B, B, C).

### 3.2.2. Learing by backpropagating action potentail (LbAP) algorithm

To train the *n*-SPSNN, we propose a local learning algorithm called learning by backpropagating action potential (LbAP) algorithm. The LbAP algorithm was inspired by physiological observations of homosynaptic plasticity dictated by backpropagating action potentials (bAPs) [16, 19]. Upon spiking at a soma, the spike propagates to the dendritic spines; this is referred to as a bAP. The bAP amplitude decays over the dendrite. However, the initial amplitude is recovered if the dendritic potential exceeds a certain threshold, indicating a bAP boost. Otherwise, the amplitude keeps decaying out. The bAP, in turn, additively perturbs the dendritic potential, such that dendritic potential above the bAP-boost threshold undergoes a large increase in potential, whereas dendritic potential below the threshold undergoes a negligible increase in potential. The key to the direction of plasticity is the calcium influx such that a large (small) influx likely induces LTP (LTD) [20-22]. Importantly, the calcium influx tends to increase with membrane potential, and thus, a bAP boost likely induces LTP, while the failure of a bAP boost likely leads to LTD [16].

The LbAP algorithm simplifies the physiological observations as follows. First, the dendritic potential at the moment of bAP arrival directly determines the plasticity direction: if the potential is above the bAP-boost threshold, the synapse gains weight, and it loses weight otherwise. Second, a delay in backpropagation is ignored, so that the weights of all relevant synapses are updated simultaneously when the postsynaptic neuron fires a spike. Therefore, the LbAP algorithm is an event-driven local algorithm, ensuring incremental learning over a learning period. To be specific, the algorithm is a postsynaptic event-driven local algorithm because the weight is renewed only upon postsynaptic events in contrast to other event-driven algorithms such as STDP rule (presynaptic and postsynaptic event-driven algorithm) [23-25] and eRBP (presynaptic event-driven algorithm) [26].

The following equation describes the LbAP algorithm:

$$\Delta w = \{\alpha H(u_d - u_{d,th}) - \beta\Theta(u_d)\}\delta(t - t_{post}) \qquad (3.1)$$

where $u_d$, $u_{d,th}$, and $H$ denote the dendritic potential at a given time, threshold for a bAP boost, and Heaviside step function, respectively. The LTP rate is determined by a positive constant $\alpha$. LTD is facilitated by a boxcar function $\Theta$ with $u_{d,th2}$ and $u_{d,th1}$ ($<u_{d,th2}$):

$$\Theta = \begin{cases} 1 \text{ if } u_{d,th1} < u_d < u_{d,th2}. \\ 0 \text{ otherwise.} \end{cases}$$

The parameter $u_{d,th1}$ denotes a threshold for LTD. The LTD rate is determined by a positive constant $\beta$. The term $\delta(t - t_{post})$ ensures a postsynaptic event-driven weight update, where $t_{post}$ refers to a postsynaptic event time. We employ weight boundaries (0 and $w_{max}$) to avoid unlimited growth of weight and switch to inhibitory synapses. The LbAP algorithm is paraphrased in pseudocode, as follows:

**function** LbAP
  **for** j $\in$ {postsynaptic spike} **do**
    **if** $u_d > u_{d,th2}$ **then** $w_{ji} \leftarrow w_{ji} + \alpha$
    **else if** $u_{d,th1} < u_d < u_{d,th2}$ **then** $w_{ji} \leftarrow w_{ji} - \beta$
    **end if**
  **end for**
**end function**.

Notably, rate-based and spike (event)-based learning schemes merge in a unified framework based on the LbAP algorithm. Regarding rate-based learning, consider two independent presynaptic neurons (N1 and N2) firing Poisson spikes at $a_1$ and $a_2$ and a postsynaptic neuron (N3) firing to the presynaptic Poisson spikes (see Fig. 3.2(a)). Applying the LbAP algorithm to the two synapses results in rate-dependent changes in weights ($w_1$ and $w_2$) with $a_1$ and $a_2$ in an unsupervised manner, as plotted in Fig. 3.2(b). The first period ($a_1 = a_2 = 25$ Hz) explains a simultaneous increase in $w_1$ and $w_2$ due to the equally high firing rates. However, the different rates in the second period ($a_1 = 25$ Hz; $a_2 = 5$ Hz) bifurcate $w_1$ and $w_2$ such that N1 with the higher rate gains weight while N2 loses weight. Alternating the rates ($a_1 = 5$ Hz; $a_2 = 25$ Hz) in the third period reverses the directions of the weight changes. Note that this learning condition recalls the monocular deprivation experiment that backs the

seminal Bienenstock–Cooper–Munro (BCM) rule [27, 28]. The result highlights rate-dependent learning (the higher the firing rate of a neuron, the more likely that the synapse with a postsynaptic neuron strengthens) in agreement with the Hebbian learning. However, unlike the basic Hebb's rule, competition between the two presynaptic neurons is induced even without explicit weight normalization, as identified by the constant sum of the weights in the second and third periods in Fig. 3.2(b). This feature highlights the key advantage of the LbAP algorithm, which enables weight normalization without access to global data unlike other normalization algorithms, e.g., heterosynaptic depression [29], Oja rule [30], and subtractive normalization [1].

The LbAP algorithm captures the temporal configuration of individual presynaptic and postsynaptic spikes. A presynaptic spike closely preceding a postsynaptic spike likely boosts a bAP at the dendritic spine, yielding LTP. In addition, using the LbAP algorithm, a pair of presynaptic and postsynaptic neurons that most frequently fire spikes in close succession (a presynaptic spike preceding a postsynaptic spike) is distinguished from the other pairs. For instance, consider a toy network of three presynaptic neurons (N1–N3) and a postsynaptic neuron (N4) in Fig. 3.2(c). One spike at a time is elicited from one of the three presynaptic neurons following a given sequence (1, 4, 2, 4, 1, 4, 3, 4) repeated 10 times (Fig. 3.2(d)). Events from N1, N2, N3, and N4 are denoted by 1, 2, 3, and 4, respectively. A supervision signal (external current) is applied to N4 to manipulate the temporal configuration of pre and postsynaptic spikes. In the sequence (1, 4, 2, 4, 1, 4, 3, 4)×10, N1 is most frequently paired with N4 (20 times) so that the synapse between N1 and N4 gains weight, whereas the other synapses undergo LTD, as shown in Fig. 3.2(e). The temporal order of spikes (a presynaptic spike preceding a postsynaptic spike in close succession) likely indicates the causality between the presynaptic postsynaptic events because a cause should precede its effect. However, the opposite order likely undermines the causality. Therefore, this example identifies the LbAP algorithm as an identifier of statistical causality between individual spikes, highlighting its suitability for spike-based learning.

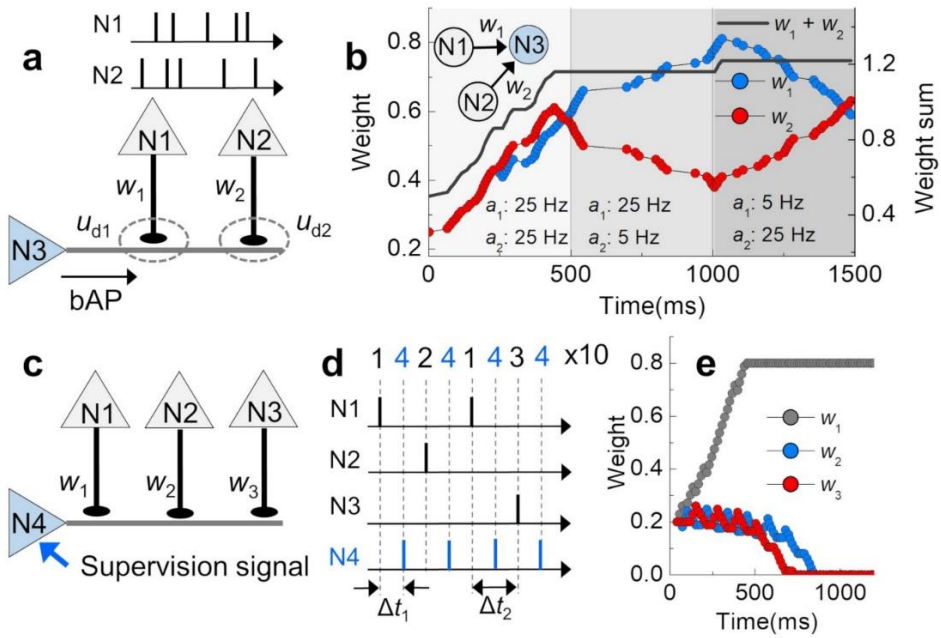


Figure 3.2. LbAP learning rule with rate and temporal codes. (a) Example of a neuronal configuration where the LbAP learning rule drives activity-dependent competition between the two presynaptic neurons N1 and N2, which share the same postsynaptic neuron N3. N1 and N2 emit Poisson spikes at activities of $a_1$ and $a_2$, respectively. (b) Evolution of weights $w_1$ and $w_2$ in response to $a_1$ and $a_2$, which differ for the three periods: 0–0.5 s, 0.5–1 s, and 1–1.5 s. The gray line denotes the sum of $w_1$ and $w_2$. (c) Example of a configuration where the LbAP learning rule drives competition between N1, N2, and N3, depending on the temporal correlation between a presynaptic and postsynaptic spike. A supervision signal applies to the postsynaptic neuron N4 to define the temporal correlation. We set $u_{d,th1}$, $u_{d,th2}$, and $w_{max}$ to 0, 1 mV, and 1, respectively. The firing threshold $u_{s,th}$ was fixed to 2.5 mV. (d) A spike sequence of (1, 4, 2, 4, 1, 4, 3, 4), where each number denotes the index of a neuron spiking at a given time. We set $\Delta t_1$ and $\Delta t_2$ to 20 ms and 30 ms, respectively. (e) Evolution of weights $w_1$, $w_2$, and $w_3$ in response to the spike sequence repeated ten times. Neuron N1 wins N2 and N3 because the unit sequence includes two 1-4 pairs, whereas both 2-4 and 3-4 pairs appear once. We set $u_{d,th1}$, $u_{d,th2}$, and $w_{max}$ to 50 μV, 1 mV, and 0.8, respectively. The neuronal parameters for both simulations are listed in Table 3.1.

### 3.2.3. Training method and capabiltiy evaluation in detail

The $n$-SPSNN ($m$-($n$×$m$)-$h$-$m$) was trained for a single-step prediction, given the $n$ previous elements in a sequence. As training data, we employed $l$-long random sequence data ($x_1$, $x_2$, …, $x_l$), where $x_i$ was randomly chosen from set $S$ (= {$s_1$, $s_2$, …, $s_m$}) with equal probability. Note that $l$ and $m$ are measures of complexity in the training data. Each element in the sequence was sampled every $\Delta t_e$ and subsequently encoded as a one-hot vector. Responding to the "1" in the one-hot vector, the input neuron in the corresponding sub-network in Fig. 3.1 fires periodic spikes at $a_0$ (=50 Hz). A supervised learning framework was used to train the $n$-SPSNN as a whole; the actual element on the present time step was considered as the correct response to the $n$ previous elements. Accordingly, the weights $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ were ad hoc updated every time step. The correct element was encoded as a one-hot vector (supervision signal) and applied to the output layer in sync with the $n$th input element of the $n$ previous elements. The supervision signal was a train of periodic current pulses at $a_0$; each pulse sufficed to evoke a spike from the neuron. Thus, periodic spikes at $a_0$ were elicited from the output neuron, which drove the update of $\boldsymbol{w}_2$. Unsupervised learning trained the weight matrix $\boldsymbol{w}_1$ because a desired spiking pattern of hidden neurons was unknown unlike training the weight matrix $\boldsymbol{w}_2$. Nevertheless, both unsupervised and supervised learning were performed within a unified framework based on the LbAP algorithm. The weight matrix $\boldsymbol{w}_1$ was loaded with random values ($0 < w_{ij} < w_{\max 1}$) initially. To avoid unwanted preset connections to the output neurons, the weight matrix $\boldsymbol{w}_2$ was loaded with constant values (0.2). Note that the lateral inhibition weights for both hidden and output layers were invariant through learning. The $n$-SPSNN was trained with the same sequence data repeatedly until saturation of the single-step prediction accuracy. The parameters in Table 3.1 were used for the simulation results, unless otherwise stated.

A single-step prediction result was determined from output neuronal spikes in a time step. Training generally hinders output spikes from multiple neurons in a given time step, and hence, the index of a single active neuron was encoded as a one-hot vector of a predicted element. Otherwise, the neuron index of the largest activity was considered to output a predicted element. The accuracy of single-step prediction was

evaluated by applying the training sequence to the *n*-SPSNN without a supervision signal and by comparing the actual output with the correct output. For instance, if an *n*-SPSNN trained with an *l*-long sequence makes correct predictions *x* times, its single-step prediction accuracy is $x/(l - n)$, where *n* is in the denominator because the first *n* elements are ignored considering the working principle of the *n*-SPSNN.

Table 3.1. Parameters for *n*-SPSNN.

| Symbol | Explanation | Value |
|---|---|---|
| $u_{\text{sth}}$ | Threshold for spikes | 10 mV |
| $u_{\text{d,th1}}$ | Threshold for LTD | 0.05 mV |
| $u_{\text{d,th2}}$ | Threshold for a bAP boost | 1 mV |
| $u_{reset}^{s}$ | Maximum hyper-polarized potential | 10 mV |
| $u_{r}^{s}$ | Rest potential at soma | 0 |
| $t_{s}^{s}$ | Postsynaptic current time constant at the soma | 15 ms |
| $t_{m}^{s}$ | Postsynaptic potential time constant at the soma | 20 ms |
| $t_{s}^{d}$ | Postsynaptic current time constant at the dendrite | 15 ms |
| $t_{m}^{d}$ | Postsynaptic potential time constant at the dendrite | 20 ms |
| $\epsilon_0$ | Pre-exponential factor | 0.0243 |
| $\kappa_0$ | Pre-exponential factor | 0.162 |
| $I^{ext}$ | An externally injected current | 1 mA |
| $w_{\text{max1}}$ | Maximum weight for sub-networks-to-hidden layer | 0.25 |
| $w_{\text{max2}}$ | Maximum weight for hidden layer-to-output layer | 0.75 |
| $\alpha$ | Synaptic permanence increment | 0.03 |
| $\beta$ | Synaptic permanence decrement | 0.03 |
| $a_0$ | Input activity | 50 Hz |
| $\Delta t_{\text{ax1}}$ | Axonal delays in synaptic chain | 100 ms |
| $\Delta t_{\text{ax2}}$ | Axonal delays for the sub-networks-to-hidden layer | 20 ms |
| $\Delta t_{\text{ax3}}$ | Axonal delays for the hidden layer-to-output layer | 20 ms |

## 3.3. Results

### 3.3.1. Sequene-prediction capacity

Fig. 3.3 compares the spiking pattern of a fully trained 20-(4×20)-40-20 SPSNN with that of an untrained SPSNN. We used a sequence of (1, 2, 3, …, 20) ($l$ = 20; $m$ = 20; $S$ = {1, 2, …, 20}), where each element was sampled every $\Delta t_e$ (=100 ms). Fig. 3.3(a) shows the response of the fully trained SPSNN to the training sequence, identifying the capability of single-step predictions, unlike the untrained SPSNN in Fig. 3.3(b). The output spikes are delayed for one time-step because of the EPSC integration rate of the used neuron model. The delay is shown in Fig. 3.3(c), where the present input element and the output spikes responding to the element on the previous time step are present on the same time step. The first $n$th elements in the training sequence cannot be predicted correctly because the $n$-SPSNN needs $n$ previous elements to predict the following element. Nevertheless, this 4-SPSNN can predict the fourth element based on the first three elements for this specific learning as in Period 1 shown in Fig. 3.3(a).

To identify the sequence-prediction capacity of the proposed $n$-SPSNN, we analyzed the prediction accuracy of an $m$-($n$×$m$)-$h$-$m$ SPSNN by varying the number of hidden neurons ($h$) and the length of a training sequence. Fig. 3.4(a) shows the measured single-step prediction accuracy with respect to $h$ for 2-, 4-, and 6-SPSNNs trained with random sequences ($l$ = 100; $m$ = 20). The data are provided in Table 3.2. The accuracy tends to increase with the number of hidden neurons until its saturation with approximately 200 hidden neurons. The accuracy for the 4-, and 6-SPSNNs reaches approximately 0.99, whereas the maximum accuracy for the 2-SPSNN is approximately 0.89. This result indicates that the number of hidden neurons is a key parameter for single-prediction capacity. Considering the negligible difference in maximum accuracy between 4- and 6-SPSNNs, $n$ is fixed to 4 hereafter. For the 2-SPSNN, the bAP-boost threshold of a hidden neuron $u^h_{\mathrm{d,th2}}$ was set to 0.5 mV (cf. a $u^h_{\mathrm{d,th2}}$ of 1 mV in Table 3.1) because two simultaneous spikes from the sub-networks fail to elevate the dendritic potential of hidden neurons above 1 mV.

The optimal number of hidden neurons offers the maximum accuracy (~1) at minimal SynOps. Fig. 3.4 reveals the rule of thumb that $h$ ($\geq 2l$) leads to an accuracy of approximately unity; therefore, $h$ ($=2l$) appears to be the optimal number. This rule of thumb is underpinned by Fig. 3.4(b), which shows the prediction accuracy of 20-(4×20)-$h$-20 SPSNNs ($h$ = 200, 500, 1000, and 2000) with respect to sequence length ($l$ = 100, 200, 500, and 1000; $m$ = 20). The rule that $h$ ($\geq 2l$) leads to the maximum accuracy (~1) holds for the data in Fig. 3.4(b). The data in Fig. 3.4(b) are provided in Table 3.3.

We trained a 6-SPSNN on the Nottingham dataset (1200 British and American folk tunes). For each tune, we used its monophonic melody only, which was discretized as 26 notes according to pitch height. The note on a given time step was encoded as a one-hot vector and input into the input layer (26 neurons). The time bin size was set to 100 ms, so that each tune was subject to periodic sampling every $\Delta t_e$ (=100 ms). This preprocessing yielded training sequences ($62 \leq l \leq 192$; $m = 26$) with heterogeneous length. The response of the 26 input neurons to a random tune is shown in Fig. 3.5(a).

To evaluate the sequence-prediction capacity, we trained a 26-(6×26)-$h$-26 SPSNN on the tunes (randomly sampled from the dataset and preprocessed as explained) by varying the number of sampled tunes. Fig. 3.5(b) shows the prediction accuracy of two SPSNNs ($h$ = 1000 and 2000) with the number of sampled tunes. For a single tune, the accuracy of both cases for one sequence is above 0.98. However, it decreases with the number of the trained sequences. For 20 sequences, the accuracy reaches approximately 0.86 for $h$ = 2000, whereas that for $h$ = 1000 is approximately 0.72.

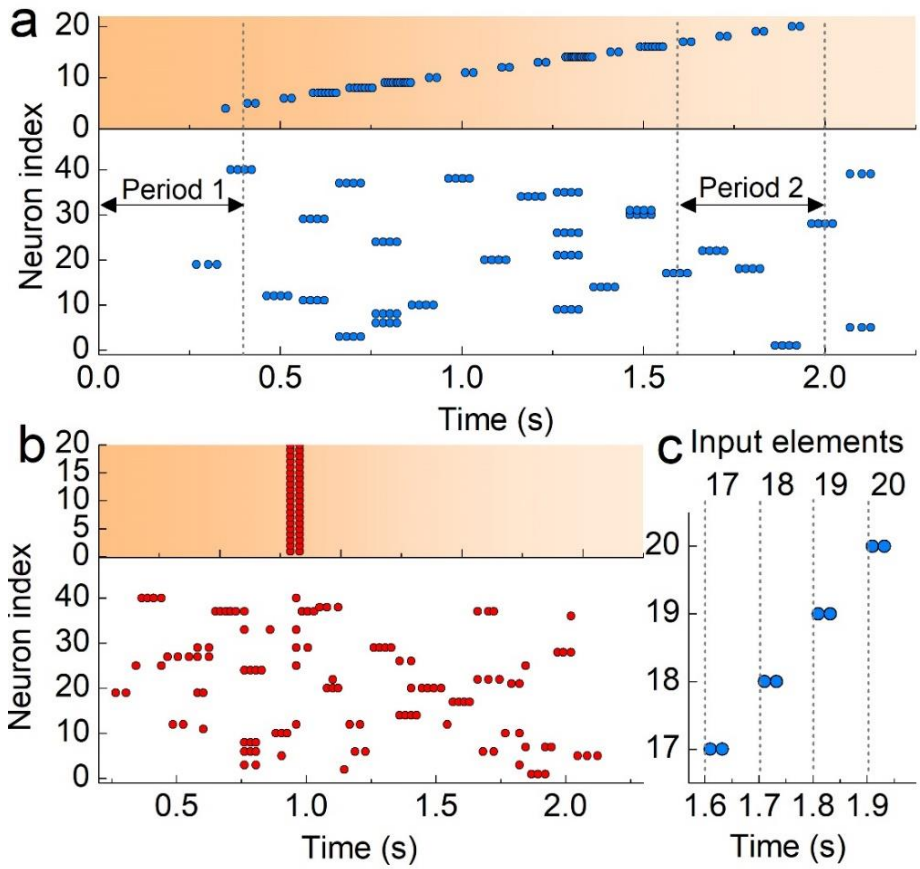


Figure 3.3. Spiking sequence before and after learning. (a) Spiking sequence of a 20-(4×20)-40-20 SPSNN for output neurons (upper panel) and hidden neurons (lower panel) in response to an input sequence of (1, 2, 3, …, 20), which is identical to training data. The spiking sequence of output neurons became associated with the training sequence in contrast to the untrained SPSNN shown in (b). (c) Spiking behavior of output neurons in Period 2 in (a), highlighting the capability of single-step predictions

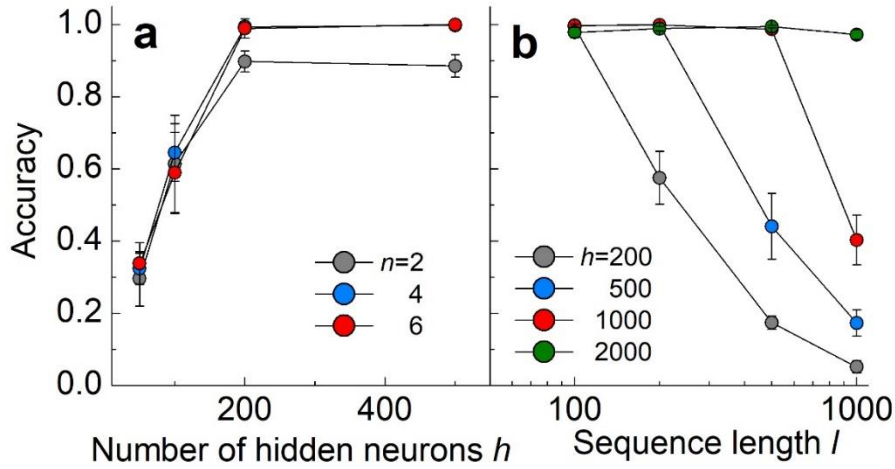


Figure 3.4. The single-step prediction accuracy. (a) Single-step prediction capability of a 20-($n$×20)-$h$-20 SPSNN with respect to the number of hidden neurons $h$ for three different $n$ values (2, 4, and 6). The SPSNN was trained using a random sequence ($l$ = 100; $m$ = 20). Each accuracy value was evaluated from ten trials; each trial includes a training period with a different random sequence and subsequent accuracy evaluation period. (b) Accuracy of a 20-(4×20)-$h$-20 SPSNN with varying training sequence length $l$ ($m$ = 20) for different $h$ values.

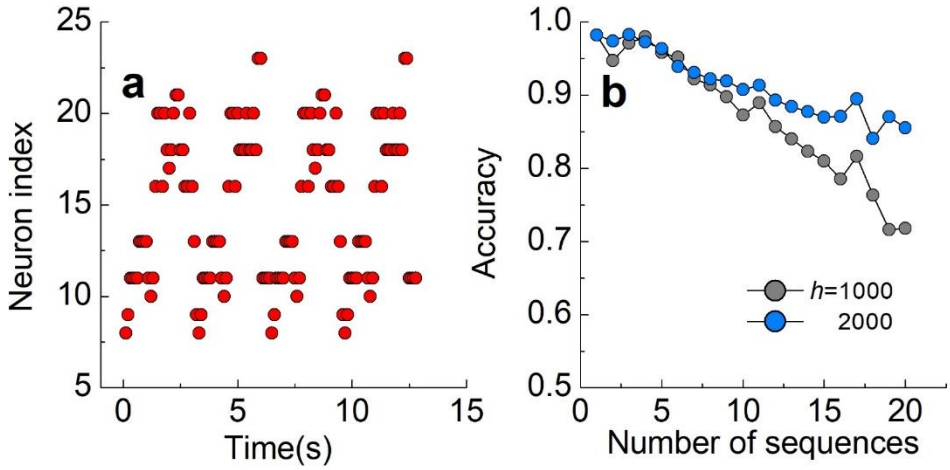


Figure 3.5. Single-step prediction accuracy on the Nottingham dataset. (a) Spiking sequence of input neurons in response to a one-hot encoded Nottingham tune. (b) Prediction-accuracy with the number of learned songs for a 26-(6×26)-*h*-26 SPSNN for different *h* values (*h* = 1000, and 2000).

Table 3.2. Single-step prediction Accuracy with respect to the number of hidden neurons *h* for three different *n* values (2, 4, and 6).

| Sequence length *l* | Network | Accuracy |
|---|---|---|
| 100 | 20-(2×20)-50-20 | 0.295 ± 0.0758 |
| 100 | 20-(2×20)-100-20 | 0.614 ± 0.135 |
| 100 | 20-(2×20)-200-20 | 0.898 ± 0.0292 |
| 100 | 20-(2×20)-500-20 | 0.886 ± 0.0307 |
| 100 | 20-(4×20)-50-20 | 0.324 ± 0.0427 |
| 100 | 20-(4×20)-100-20 | 0.646 ± 0.0798 |
| 100 | 20-(4×20)-200-20 | 0.994 ± 0.00728 |
| 100 | 20-(4×20)-500-20 | 0.998 ± 0.00439 |
| 100 | 20-(6×20)-50-20 | 0.338 ± 0.0578 |
| 100 | 20-(6×20)-100-20 | 0.589 ± 0.112 |
| 100 | 20-(6×20)-200-20 | 0.989 ± 0.0270 |
| 100 | 20-(6×20)-500-20 | 1 |

Table 3.3. The single-step prediction accuracy with varying training sequence length $l$ ($m$ = 20) for different $h$ values.

| Sequence length $l$ | Network | Accuracy |
|---|---|---|
| 100 | 20-(4×20)-200-20 | 0.994 ± 0.00728 |
| 100 | 20-(4×20)-500-20 | 0.998 ± 0.00439 |
| 100 | 20-(4×20)-1000-20 | 0.997 ± 0.00503 |
| 100 | 20-(4×20)-2000-20 | 0.978 ± 0.0143 |
| 200 | 20-(4×20)-200-20 | 0.576 ± 0.0733 |
| 200 | 20-(4×20)-500-20 | 0.999 ± 0.00215 |
| 200 | 20-(4×20)-1000-20 | 0.999 ± 0.00215 |
| 200 | 20-(4×20)-2000-20 | 0.989 ± 0.00627 |
| 500 | 20-(4×20)-200-20 | 0.174 ± 0.0184 |
| 500 | 20-(4×20)-500-20 | 0.441 ± 0.0913 |
| 500 | 20-(4×20)-1000-20 | 0.987 ± 0.00512 |
| 500 | 20-(4×20)-2000-20 | 0.995 ± 0.00343 |
| 1000 | 20-(4×20)-200-20 | 0.0519 ± 0.0168 |
| 1000 | 20-(4×20)-500-20 | 0.174 ± 0.0368 |
| 1000 | 20-(4×20)-1000-20 | 0.403 ± 0.0689 |
| 1000 | 20-(4×20)-2000-20 | 0.972 ± 0.0131 |

### 3.3.2. Associative recall (sequence-to-sequence prediction)

The high accuracy of single-step prediction of the *n*-SPSNN offers the basis for associative recall (sequence-to-sequence prediction). For associative recall, the *n*-SPSNN architecture is modified such that feedback from the output to the input layer is employed to pass the prediction result on to the input. An advantage is that the output result (one-hot vector) can be applied to the input layer without additional encoding. To identify associative recall capability, we repeatedly trained a 20-(4×20)-40-20 SPSNN without feedback using the sequence (1, 2, 3, …, 20) ($l$ = 20; $m$ = 20). An associative recall test with the feedback followed every training epoch; associative recall was triggered by applying the first four elements of the sequence. Fig. 3.6 shows the progress of associative recall with the repetition of training. The 20-(4×20)-40-20 SPSNN eventually succeeds in recalling the whole sequence after repeating training four times.

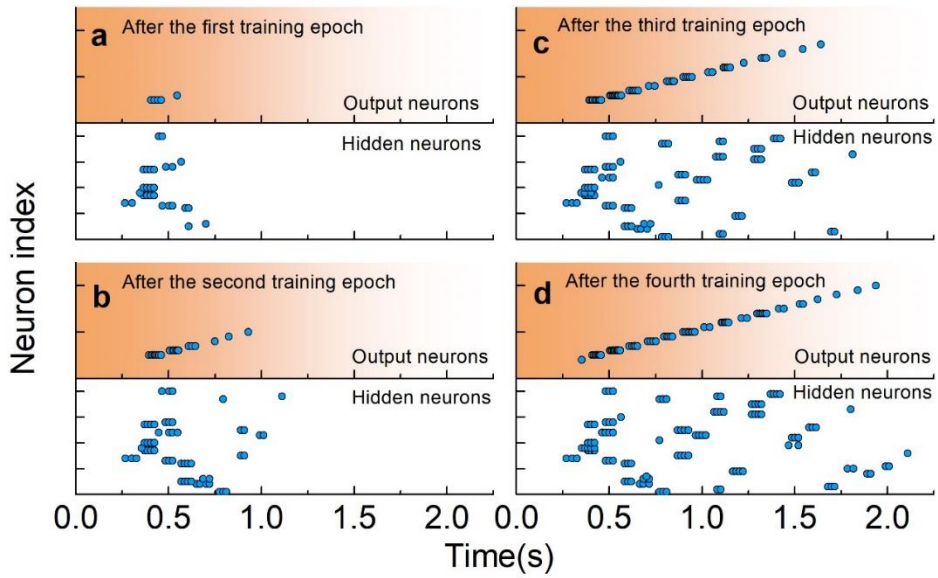


Figure 3.6. Associative recall. Associative recall capability of a 20-(4×20)-40-20 SPSNN after the (a) first, (b) second, (c) third, and (d) fourth training epoch. The training sequence was (1, 2, 3, …, 20) ($l$ = 20; $m$ = 20; $\Delta t_e$ = 100 ms). For each case, the associative memory was triggered by the initial four elements (1, 2, 3, 4) of the total sequence.

### 3.3.3. Robustness of learning and inference to variability in sequence

Considering that real-world sequences include many imperfections, e.g., typo and noise, sequence-learning hypotheses need to make correct predictions despite the presence of imperfections. In this regard, the robustness of the *n*-SPSNN to errors in input encoding was examined. A 4-SPSNN [20-(4×20)-200-20] was trained using a random sequence ($l$ = 100; $m$ = 20), and its single-step prediction accuracy was measured with a test sequence that is identical to the training sequence but with a few different elements from the training sequence. They were chosen randomly. The different elements indicate errors in input encoding; their number $x$ defines the error rate as $x/l$. We evaluated the average prediction accuracy for a given error rate in the range 0–0.25 on 20 trials. Fig. 3.7(a) shows a linear decrease in accuracy with error rate, reaching approximately 0.62 at the maximum error rate (0.25). The results are compared with the error-tolerance of an LSTM and GRU, which are state-of-the-art sequence learning hypotheses. The LSTM and GRU used for this comparative study are elaborated in Appendix 3.5. Similar to the 4-SPSNN, the LSTM and GRU undergo the degradation of prediction accuracy with error rate. However, their degradation rates are faster than that of the 4-SPSNN, insomuch as the accuracy for the LSTM and GRU reaches approximately 0.54 and 0.53, respectively, with an error rate of 0.25 (Fig. 3.7(a)). This comparison ensures a large tolerance of encoding error for the *n*-SPSNN trained with the LbAP algorithm compared to the state-of-the-art sequence learning hypotheses.

Prediction-robustness to variability in input-encoding delay is the key to the application to asynchronous neuromorphic hardware. To identify this robustness, a 20-(4×20)-200-20 SPSNN was trained using a sequence ($l$ = 100; $m$ = 20) with constant $\Delta t_e$ (=100 ms), and its single-step prediction accuracy was investigated with the same sequence but with randomly varying $\Delta t_e$ over the sequence. The delay in input-encoding $\Delta t_e'$ was sampled from a Gaussian distribution function, which is centered at $\Delta t_e$ (=100 ms) with a standard deviation of $\sigma$, i.e., $\Delta t_e' \sim N(\Delta t_e, \sigma)$. The delay was sampled for every interval over the test sequence. The standard deviation $\sigma$ is a measure of the variability in input-encoding delay. The measured prediction accuracy with the standard deviation is shown in Fig. 3.7(b). The accuracy tends to

decrease with the standard deviation because the difference in input-encoding delay between the training and test sequences becomes larger with the standard deviation. Nevertheless, an accuracy of approximately 0.77 is maintained even with a standard deviation of 25 ms (25% of the center value). All data in Fig. 3.7 are provided in Table 3.4.

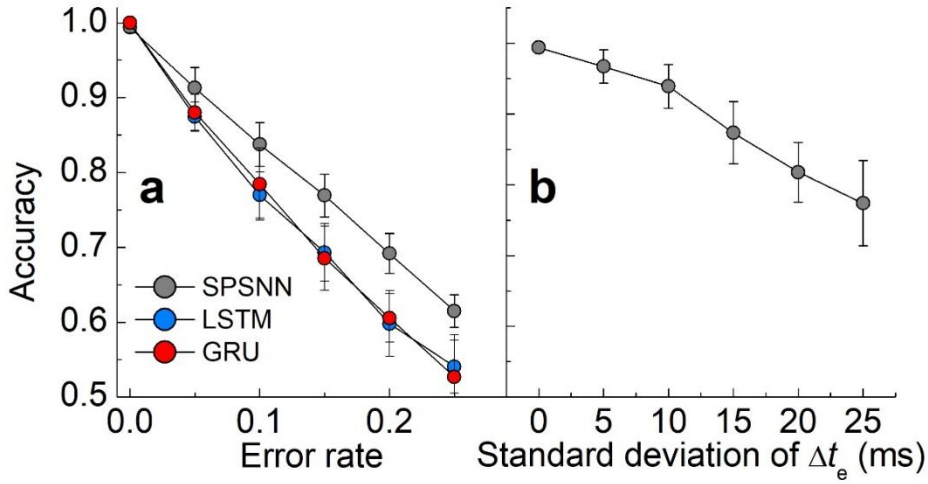


Figure 3.7. Tolerance of the SPSNN to errors in input sequences. (a) Robustness of single-step prediction for a 20-(4×20)-200-20 SPSNN to variability in elements in an input sequence ($l$ = 100; $m$ = 20) in comparison with LSTM and GRU. (b) Degradation of prediction accuracy for the same SPSNN with respect to variability in input sampling period ($\Delta t_e$).

Table 3.4. Tolerance to errors in input sequences and variability in input sampling period.

| Error rate | Sequence length $l$ | Network | | |
|---|---|---|---|---|
| | | 20-(4×20)-200-20 SPSNN | LSTM | GRU |
| 0 | 100 | 0.994 ± 0.00728 | 1 | 1 |
| 0.05 | 100 | 0.913 ± 0.0275 | 0.874 ± 0.0196 | 0.880 ± 0.0240 |
| 0.1 | 100 | 0.838 ± 0.0294 | 0.770 ± 0.0306 | 0.784 ± 0.0479 |
| 0.15 | 100 | 0.769 ± 0.0287 | 0.693 ± 0.0383 | 0.685 ± 0.0427 |
| 0.2 | 100 | 0.692 ± 0.0267 | 0.598 ± 0.0441 | 0.606 ± 0.0323 |
| 0.25 | 100 | 0.615 ± 0.0218 | 0.541 ± 0.0353 | 0.527 ± 0.0562 |
| s.d. of $\Delta t_e$ (ms) | | | | |
| 0 | 100 | 0.994 ± 0.00728 | - | - |
| 5 | 100 | 0.967 ± 0.0235 | - | - |
| 10 | 100 | 0.939 ± 0.0307 | - | - |
| 15 | 100 | 0.873 ± 0.0440 | - | - |
| 20 | 100 | 0.818 ± 0.0421 | - | - |
| 25 | 100 | 0.774 ± 0.0605 | - | - |

### 3.3.4. Learning efficiency

Energy-efficient learning is an important attribute of a learning algorithm embedded in neuromorphic hardware [5, 12]. In this regard, a high learning rate is beneficial to energy-efficient learning, reducing the number of operations that significantly consume power. The SynOps is such an operation, which indicates a single update on a neuronal membrane potential upon an event. Therefore, the number of SynOps required for successful learning is a direct measure of energy-efficiency in learning. This quantity was evaluated for a 20-(4×20)-2000-20 SPSNN learning sequences of different lengths ($20 \leq l \leq 1000$; $m = 20$). Success in learning was defined by prediction accuracy above 0.97, and hence, the iterative training terminated when an accuracy of 0.97 was reached. The results are plotted in Fig. 3.8(a). The number of SynOps increases with the sequence length because a longer training sequence needs more ad hoc updates over the whole sequence, which inevitably increases SynOps.

We compared the required number of SynOps for successful learning with the required number of multiply-accumulate (MAC) operations for an LSTM and GRU. As for the SPSNN, both LSTM and GRU were trained using sequences of different lengths ($20 \leq l \leq 1000$; $m = 20$), and the training terminated when the single-step prediction accuracy reached 0.97. Details of the LSTM and GRU are provided in Appendix 3.5. The evaluation results are co-plotted in Fig. 3.8(a), highlighting the efficient learning for the SPSNN with approximately two orders of magnitude fewer energy-consuming operations. The efficiency in learning is attributed to the fast learning rate facilitated by the LbAP algorithm, which is identified by monitoring the evolution of prediction accuracy with the number of training iterations (epochs). As shown in Fig. 3.8(b), a 20-(4×20)-2000-20 SPSNN trained using a random sequence ($l = 1000$; $m = 20$) achieves its maximum accuracy (~0.98) in five training iterations, while the LSTM and GRU needs approximately two orders of magnitude more iterations. This comparison indicates the fast learning rate of the LbAP algorithm. In the SPSNN, hidden neurons compete through lateral inhibition and work as unique subsequence detectors. This unsupervised learning with winner-takes-all results in fast learning speed. Moreover, the learning rate is independent of

the network size (here, the number of hidden neurons) and sequence length $l$ as shown in the comparison with a 20-(4×20)-20-20 SPSNN trained using a random sequence ($l = 10$; $m = 20$) (Fig. 3.8(b)). The smaller network could learn the sequence with four iterative training steps, identifying a non-scaling learning rate with both network size and sequence length.

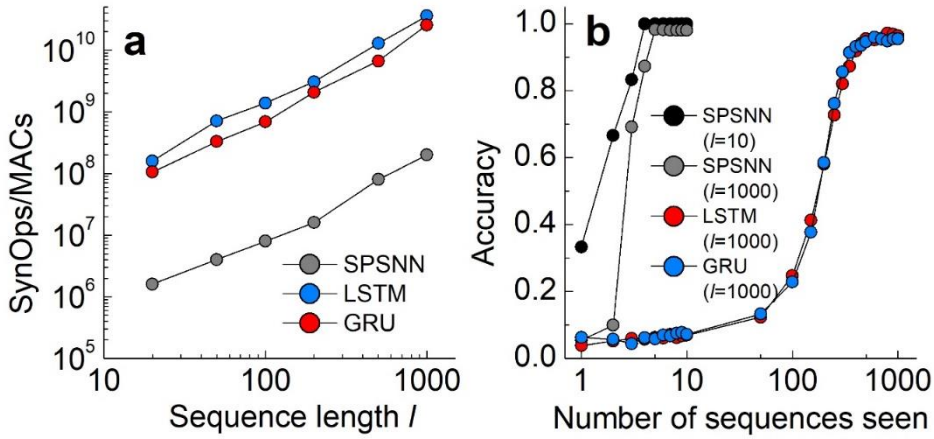


Figure 3.8. Efficiency in learning. (a) Number of SynOps for a 20-(4×20)-2000-20 SPSNN until a prediction accuracy of 0.97 with respect to sequence length ($20 \leq l \leq 1000$; $m$ = 20). LSTM and GRU are compared with the SPSNN in terms of the number MAC operations required to reach the same prediction accuracy (0.97). (b) Single-step prediction accuracy evolution for a 20-(4×20)-2000-20 SPSNN, LSTM, and GRU with the number of training iterations. They were trained using random sequences ($l$ = 1000; $m$ = 20). For comparison, the same data for a 20-(4×20)-20-20 SPSNN trained using a random sequence ($l$ = 10; $m$ = 20) are co-plotted.

## 3.4. Conclusion

We proposed an SNN architecture suitable for single-step prediction given $n$ previous elements in a training sequence, referred to as $n$-SPSNN. The key to the $n$th order sequence prediction is the sub-networks of synaptic chains that serve as working memory. This $n$-SPSNN architecture can learn sequences of various lengths using the LbAP algorithm as a unified learning framework. The LbAP algorithm is a postsynaptic event-driven learning algorithm of locality; each synapse involves a single local state variable (dendritic potential) so that memory usage is minimal. The competition between synapses with the same postsynaptic neuron is facilitated by the LbAP algorithm, which realizes effective weight normalization using local state variables only. The LbAP algorithm endows the $n$-SPSNN with the capabilities of single-step prediction and associative recall.

The sequence prediction robustness to variability in the test sequence element highlights its high tolerance to errors in input encoding, which is higher than the state-of-the-art sequence learning hypotheses LSTM and GRU. The $n$-SPSNN also offers the sequence prediction robustness to variability in intervals between neighboring elements, implying high tolerance to random changes in input-encoding delay. The efficiency in learning is another advantage of the $n$-SPSNN with the LbAP algorithm. The learning is completed in a few iterations. The iteration number necessary for success in learning hardly scales with the network size and sequence length; therefore, the LbAP algorithm can train large-scale SNNs in an energy- and time-efficient manner.

Nevertheless, the learning capacity of the $n$-SPSNN is limited mainly by (i) the use of one-hot coding for input (extremely sparse coding) and (ii) the limited number of hidden neurons $h$. The former limits the number of symbol representations for a given network setting. Therefore, dense coding is desired to improve the learning capacity of a given $n$-SPSNN, which we leave as a future work for the moment. Considering the latter, the optimal number of hidden neurons $h$ for successful learning scales with sequence length $l$ such that $h \approx 2l$. The number of learnable sequences with different lengths is also determined by this rule; the entire length of

the concatenated sequences should satisfy this rule. Therefore, the network should be preset appropriately considering the complexity of the sequences that the *n*-SPSNN is trained on.

## 3.5. Appendix

### Appendix I. Multi-compartment neuron model

Multi-compartment neurons were employed in the hidden and output layers in the *n*-SPSNN; each neuron is with a soma and multiple dendritic spines. Accordingly, somatic and dendritic potentials were evaluated separately. The somatic potential of neuron *i* ($u_i^s$) was evaluated using the SRM [2] expressed as

$$u_i^s(t) = \eta(t - \hat{t}_i) + \sum_j w_{ij} \sum_f \epsilon(t - t_j^{(f)}) + \int_0^\infty \kappa(s) I_i^{ext}(t - s) ds, \quad (3.2)$$

where $\hat{t}_i$, $w_{ij}$, and $t_j^{(f)}$ denote the last spike time of neuron *i*, the weight of the synapse between neurons *j* and *i*, and the $f^{\text{th}}$ spike time of neuron *j*, respectively. A refractory period and leaky integration of postsynaptic current are realized by the kernels $\eta$ and $\epsilon$, respectively. An externally injected current into neuron *i* for supervised learning is denoted by $I_i^{ext}$.

$$\eta(t) = -(u_{reset}^s - u_r^s)\exp(-\frac{t}{t_m^s})\Theta(t) \quad (3.3)$$

$$\epsilon(t) = \epsilon_0 \left[\exp\left(-\frac{t}{t_m^s}\right) - \exp(-\frac{t}{t_s^s})\right]\Theta(t), \quad (3.4)$$

$$\kappa(t) = \kappa_0 \exp\left(-\frac{t}{t_m^s}\right)\Theta(t), \quad (3.5)$$

where $u_{reset}^s$ and $u_r^s$ are the most hyperpolarized membrane potential (immediately after spiking) and the resting potential at the soma, respectively. At the soma, the postsynaptic current and potential decay exponentially with time constants of $t_s^s$ and $t_m^s$, respectively. The pre-exponential factors $\epsilon_0$ and $\kappa_0$ are positive constants. The somatic membrane potential exceeding a threshold for spiking fires a spike, and the potential is evaluated on the next time step with the updated $\hat{t}_i$.

The SRM applied to the dendritic potential evaluation. However, because no dendritic spikes are allowed, the first term on the right-hand side of (3.2) is ruled out. Furthermore, because supervision current pulses are applied to the soma only, the last term on the right-hand side of (3.2) is excluded. The same kernel in (3.4) was used but with the parameters $t_s^d$ and $t_m^d$ instead of $t_s^s$ and $t_m^s$. The replacement considers different responses of postsynaptic current and membrane potential to presynaptic spikes for a soma and dendritic spine, based on physiological observations [16, 31]. The neuronal parameters used in this study are listed in Table 4.1.

**Appendix II. Training RNN with LSTM and GRU layer**

For the LSTM and GRU experiment, we trained a two-layer neural network with a recurrent unit. The first layer is the LSTM or GRU layer with 40 units and the second layer is a dense layer with 20 output neurons. Training employed categorical cross-entropy as a loss function and the Adam optimizer with a learning rate of 0.001. To realize *n*th order prediction, *n*-long subsequences were taken as inputs and encoded as an *m*-long real-valued vector (0–1) using a real-valued dense distributed representation. The output was an *m*-long vector that indicates a predicted element given a subsequence including *n* preceding elements. During training, a desired output was encoded as a one-hot vector with which the weights were updated ad hoc, i.e., online learning.

## 3.6. Bibliography


[1] P. Dayan and L. F. Abbott. London: The MIT Press, 2001.

[2] W. Gerstner and W. M. Kistler. Cambridge University Press, 2002.

[3] D. S. Jeong, *Journal of Applied Physics,* vol. 124, no. 15, p. 152002, 2018.

[4] M. Pfeiffer and T. Pfeil, *Frontiers in neuroscience,* vol. 12, p. 774, 2018.

[5] E. O. Neftci, *iScience,* vol. 5, pp. 52-68, 2018.

[6] M. Davies *et al.*, *IEEE Micro,* vol. 38, no. 1, pp. 82-99, 2018.

[7] P. A. Merolla *et al.*, *Science,* vol. 345, no. 6197, pp. 668-673, August 8, 2014.

[8] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, *Neural Netw.,* vol. 111, pp. 47-63, 2019.

[9] P. O'Connor, D. Neil, S.-C. Liu, T. Delbruck, and M. Pfeiffer, *Front. Neurosci.,* Original Research vol. 7, no. 178, 2013.

[10] H. Mostafa, *IEEE Trans. Neural Netw. Learn. Syst.,* vol. 29, no. 7, pp. 3227-3235, 2018.

[11] S. K. Esser *et al.*, *Proc. Natl. Acad. Sci. U. S. A.,* vol. 113, no. 41, pp. 11441-11446, 2016.

[12] V. Kornijcuk and D. S. Jeong, *Advanced Intelligent Systems,* 2019.

[13] E. O. Neftci, C. Augustine, S. Paul, and G. Detorakis, *Front. Neurosci.,* Original Research vol. 11, p. 324, 2017-June-21 2017.

[14] S. Hochreiter and J. Schmidhuber, *Neural computation,* vol. 9, no. 8, pp. 1735-1780, 1997.

[15] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, *arXiv preprint arXiv:1412.3555,* 2014.

[16] P. J. Sjöström and M. Häusser, *Neuron,* vol. 51, no. 2, pp. 227-238, 2006.

[17] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, *Proc. IEEE,* vol. 102, no. 5, pp. 652-665, 2014.

[18] N. Q. S. Moradi, F. Stefanini, G. Indiveri, *IEEE Trans. Biomed. Circuits Syst.,* vol. 12, pp. 106-122, 2018.

[19] P. J. Sjöström, G. G. Turrigiano, and S. B. Nelson, *Neuron,* vol. 32, no. 6, pp. 1149-1164, 2001.

[20] J. Lisman, *Proc. Natl. Acad. Sci.,* vol. 86, no. 23, pp. 9574-9578, December 1, 1989.

[21] C. Hansel, A. Artola, and W. Singer, *European Journal of Neuroscience,* vol. 9, no. 11, pp. 2309-2322, 1997.

[22] K. Cho, J. P. Aggleton, M. W. Brown, and Z. I. Bashir, *J. Physiol.,* vol. 532, no. Pt 2, pp. 459-466, 2001.

[23] S. Song, K. D. Miller, and L. F. Abbott, *Nat. Neurosci.,* 10.1038/78829 vol. 3, no. 9, pp. 919-926, 2000.

[24] R. C. Froemke and Y. Dan, *Nature,* 10.1038/416433a vol. 416, no. 6879, pp. 433-438, 2002.

[25] E. M. Izhikevich and N. S. Desai, *Neural Comput.,* vol. 15, no. 7, pp. 1511-1523, 2003.

[26] E. Neftci, S. Das, B. Pedroni, K. Kreutz-Delgado, and G. Cauwenberghs, *Front. Neurosci.,* Original Research vol. 7, no. 272, 2014.

[27] E. Bienenstock, L. Cooper, and P. Munro, *J. Neurosci.,* vol. 2, no. 1, pp. 32-48, 1982.

[28] L. N. Cooper and M. F. Bear, *Nat. Rev. Neurosci.,* 10.1038/nrn3353 vol. 13, no. 11, pp. 798-810, 2012.

[29] I. R. Fiete, W. Senn, C. Z. H. Wang, and R. H. R. Hahnloser, *Neuron,* vol. 65, no. 4, pp. 563-576, 2010.

[30] E. Oja, *Journal of Mathematical Biology,* vol. 15, no. 3, pp. 267-273, 1982.

[31] J. C. Magee, *Nat Rev Neurosci,* 10.1038/35044552 vol. 1, no. 3, pp. 181-190, 2000.

# 4. eWB: Event-based weight binarization algorithm for spiking neural networks

## 4.1. Introduction

There has been growing interest in fast, efficient, and compact neuromorphic computing for high-performance processing of large amounts of data for on-chip learning. Spiking neural networks (SNNs) are a promising model for energy-efficient neuromorphic computing [1-3]. Their energy efficiency is mainly due to the sparse event-based asynchronous data processing and learning weights, as opposed to the case for deep neural networks (DNNs), which utilize error-backpropagation algorithms (BP) for layer-wise synchronous weight updates in dedicated learning phases [1]. Further efficiency improvements are gained when SNNs are implemented on dedicated neuromorphic hardware [4, 5]. To date, several event-based learning algorithms have been proposed, including STDP [6-8], eRBP [9], sequence-predicting SNN [10], ReSuMe [11], tempotron [12], and Spikeprop [13]. However, because most of these event-based algorithms use multi-bit weights, their hardware implementation requires large on-chip memory capacity and intensive computing power, which degrades their energy efficiency. Weight quantization has been considered to address this issue, for example, in recent STDP-based algorithms with quantized weights [7, 8, 14-16]. However, achieving a competitive classification accuracy commonly requires (i) a large number of trainable parameters, especially those related to hidden neurons, (ii) an inhomogeneous learning framework to consider BP and STDP separately, and (iii) multi-bit weights for output evaluation.

Learning binary weights is an extreme case of weight quantization. The use of 1-bit weights significantly reduces on-chip memory usage considering the $O(n^2)$ memory complexity of synapses. Additionally, the leaky integrate-and-fire (LIF) model and its variations involve the multiplication of weights and low-pass filtered spikes [17]. Thus, learning binary weights avoids multiplication and significantly reduces power consumption and processing time.

To this end, we propose an event-driven weight binarization (eWB) algorithm to

learn binary weights (-1, 1) in an event-based manner. The eWB algorithm uses the Lagrange multiplier method (LMM) based on a Lagrange function that combines a conventional loss function and constraints on binary weights. Each synapse is given a binary weight constraint function and a Lagrange multiplier. The binarization of each weight is independent of the variables in the other synapses. This ensures the locality of eWB. Because a conventional loss function is also used, eWB is not a standalone learning algorithm. Instead, it requires an additional event-based supervised learning algorithm based on a loss function such as eRBP. As a proof of concept, we combine eWB and eRBP (eWB-eRBP) to train fully connected multilayer SNNs on MNIST. The results demonstrate successful weight binarization at the cost of an accuracy reduction by approximately 1.85%.

## 4.2. eWB algorithm

### 4.2.1. Lagrange multiplier method

LMM is a strategy to solve general nonlinear nonlinear programming problems (NLPs) [19]. An NLP is an optimization problem whose optimal solution is determined by constraints in conjunction with a nonlinear objective function. In the minimization problem, a general continuous equality-constrained NLP can be stated as

minimize $l(\boldsymbol{w})$; $\boldsymbol{w} = [w_1, w_2, \cdots, w_n]$

subject to $\boldsymbol{g}(\boldsymbol{w}) = 0$; $\boldsymbol{g} = [g_1, g_2, \cdots, g_m]$.

The LMM calculates the local maxima or minima of the objective function within the given equality constraints. The Lagrange function $L$ for the objective function $l$ and constraints $\boldsymbol{g}$ is expressed as

$$L(\boldsymbol{w}, \boldsymbol{\lambda}) = l(\boldsymbol{w}) + \boldsymbol{\lambda} \cdot \boldsymbol{g}(\boldsymbol{w}), \tag{4.1}$$

where $\boldsymbol{\lambda}(=[\boldsymbol{\lambda_1}, \boldsymbol{\lambda_2}, \cdots, \boldsymbol{\lambda_m}])$ is a vector of Lagrange multipliers with one multiplier for each of the $m$ constraints. If $\boldsymbol{w}^*$ is a local extremum point of the objective function $l(\boldsymbol{w})$ subject to $\boldsymbol{g}(\boldsymbol{w}) = \mathbf{0}$, the following equalities hold:

$$\begin{cases} \nabla_{\boldsymbol{w}} l(\boldsymbol{w}) + \boldsymbol{\lambda} \cdot \nabla_{\boldsymbol{w}} \boldsymbol{g}(\boldsymbol{w}) = \mathbf{0} \text{ at } \boldsymbol{w}^* \\ \boldsymbol{g}(\boldsymbol{w}^*) = \mathbf{0} \end{cases} \tag{4.2}$$

This optimal solution $\boldsymbol{w}^*$ can be calculated from the gradient of Eq. 4.1:

$$\boldsymbol{\nabla}_{\boldsymbol{w},\boldsymbol{\lambda}} \boldsymbol{L}(\boldsymbol{w}, \boldsymbol{\lambda}) = \boldsymbol{\nabla}_{\boldsymbol{w}}[l(\boldsymbol{w}) + \boldsymbol{\lambda} \cdot \boldsymbol{g}(\boldsymbol{w})] + \boldsymbol{\nabla}_{\boldsymbol{\lambda}}[l(\boldsymbol{w}) + \boldsymbol{\lambda} \cdot \boldsymbol{g}(\boldsymbol{w})].$$

The condition $\nabla_{\boldsymbol{w},\boldsymbol{\lambda}} L(\boldsymbol{w}, \boldsymbol{\lambda}) = 0$ is equivalent to the following conditions:

$$\begin{cases} \boldsymbol{\nabla}_{\boldsymbol{w}}[l(\boldsymbol{w}) + \boldsymbol{\lambda} \cdot \boldsymbol{g}(\boldsymbol{w})] = \boldsymbol{\nabla}_{\boldsymbol{w}} l(\boldsymbol{w}) + \boldsymbol{\lambda} \cdot \boldsymbol{\nabla}_{\boldsymbol{w}} \boldsymbol{g}(\boldsymbol{w}) = \mathbf{0} \\ \boldsymbol{\nabla}_{\boldsymbol{\lambda}}[l(\boldsymbol{w}) + \boldsymbol{\lambda} \cdot \boldsymbol{g}(\boldsymbol{w})] = \boldsymbol{g}(\boldsymbol{w}) = \mathbf{0} \end{cases},$$

which are identical to the conditions in Eq. 4.2. Therefore, the optimal point $\boldsymbol{w}^*$ leads to

$$\nabla_{\boldsymbol{w},\boldsymbol{\lambda}} L(\boldsymbol{w}, \boldsymbol{\lambda}) = 0. \tag{4.3}$$

The solution to Eq. 4.3 can be calculated using a basic differential multiplier method [20], in which the optimal $\boldsymbol{w}$ and $\boldsymbol{\lambda}$ are calculated using a gradient descent and ascent method, respectively, i.e.,

$$\begin{cases} \boldsymbol{w}^{k+1} = \boldsymbol{w}^k - \eta_w \nabla L_{\boldsymbol{w}}(\boldsymbol{w}^k, \boldsymbol{\lambda}^k) \\ \boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k + \eta_\lambda \boldsymbol{g}(\boldsymbol{w}^k) \end{cases} \tag{4.4}$$

where $\eta_w$ and $\eta_\lambda$ are learning rates. Note that this method is compatible with the BP in DNNs if the model parameters $\boldsymbol{w}$ are updated using a gradient descent method but with the Lagrange function $L$ rather than the loss function $l$ taken as the objective function.

### 4.2.2. eWB algorithm

The eWB algorithm is based on LMM with binary weight constraints. The key feature of this algorithm is that the configuration of binary weights over the network is subject to optimization, unlike common weight binarization methods that force the weights to assume binary values using particular binarization functions. Because each synapse is given a binary weight constraint and a Lagrange multiplier, the total numbers of constraints and Lagrange multipliers are equal to the number of synapses in the SNNs. Here, we select the binary weight constraint g for a given synapse as

$$g_i(w_i, t) = \left(1 - w_i^2\right) s_i(t), \tag{4.5}$$

which is zero if $w_i = \pm 1$. We introduce the spike function $s_i$, which yields one when a presynaptic (or postsynaptic) spike occurs and zero otherwise, and hence enables event-based asymptotic binarization. The loss function $l(\boldsymbol{w},t)$ should be chosen to enable event-based weight updates and satisfy

$$\frac{\partial l}{\partial w_i} = h(\boldsymbol{w}, t) s_i(t)$$

where $h$ is the product of the backpropagating error and the derivative of postsynaptic output. The Lagrange function $L$ is given by

$$L(\boldsymbol{w}, \boldsymbol{\lambda}, t) = l(\boldsymbol{w}, t) + \sum_i \lambda_i \left(1 - w_i^2\right) s_i(t)$$

Consequently, the weight and Lagrange multiplier are updated upon a presynaptic (or postsynaptic) spike of synapse $w_i$ conforming to Eq. 4.4:

$$\begin{cases} w_i \leftarrow w_i - \eta_w \dfrac{\partial L}{\partial w_i} \\ \lambda_i \leftarrow \lambda_i + \eta_\lambda \dfrac{\partial L}{\partial \lambda_i} \end{cases} \tag{4.6}$$

These real-valued weights and Lagrange multipliers are stored in the memory for successive event-based updates. However, during training with eWB, both signal

forward propagation and error backpropagation use the forced-to-be-binary weights $w^{\mathrm{b}}$:

$$w_i^{\mathrm{b}} = \mathrm{Sign}(w_i) = \begin{cases} +1 \; if \; w_i \geq 0 \\ -1 \; \text{otherwise} \end{cases} \tag{4.7}$$

This avoids the multiplication of real-valued weights and low-pass filtered spikes in the LIF model, thereby significantly reducing the computational complexity and, thus, the power consumption.

### 4.2.3. eWB-eRBP algorithm

As a proof of concept, we chose eRBP for combination with eWB (eWB-eRBP). As shown in Fig. 4.1, eRBP is a three-factor rule based on (i) presynaptic events, (ii) approximated gradients of postsynaptic activation, and (iii) error signals through random feedback channels [9]. The eRBP algorithm is elaborated in the Appendix 4.6. Accordingly, the loss function $l$ is considered to be the mean-squared difference between the target and actual outputs. The eRBP algorithm defines the derivative of the loss function with respect to the weight $w_{ij}$ between the presynaptic neuron $j$ and postsynaptic neuron $i$ as

$$\frac{\partial l}{\partial w_{ij}} \stackrel{\text{def}}{=} T_i(t)\Theta(I_i)s_j^{pre}(t), \tag{4.8}$$

where $T_i$ is a random backpropagation error for the weight update. The gradient of the postsynaptic activation is approximated by a boxcar function $\Theta$ with two transition points ($b_{\min}$ and $b_{\max}$), which is a function of the synaptic current $I_i$:

$$\Theta(I_i) = \begin{cases} 1 \text{ if } b_{min} < I_i < b_{max} \\ 0 \text{ otherwise} \end{cases}. \tag{4.9}$$

The spike train of the presynaptic neuron $j$ is denoted by $s_j^{pre}$. This term allows weight update upon the occurrence of presynaptic events only; therefore, eRBP is a presynaptic event-based learning algorithm. We tailor the binary weight constraint function $g$ to eRBP such that

$$g(w_{ij}) = (1 - w_{ij}^2)\Theta(I_i)s_j^{pre}(t). \tag{4.10}$$

The modification can be seen by comparing with Eq. 4.5. For compatibility with eRBP, eWB is also assumed to be driven by presynaptic events. Additionally, we incorporate the approximated postsynaptic activation gradient Θ into the constraint function. Θ is included in the constraint function to synchronize the weight update for reducing the constraint function *g* with the update for reducing the loss function *l*. Otherwise, it may be possible that only the constraint *g* is reduced during the weight update, irrespective of the loss function *l*, especially when Θ = 0.

The derivative of the boxcar function Θ in Eq. (9) is zero except for the two transition points ($b_{\min}$ and $b_{\max}$), which are singular points. Thus, the boxcar function is non-differentiable. As a workaround, we assume that the synaptic current $I_i$ avoids these transition points when presynaptic events occur, which is highly probable because the probability of the current being equal to either of the two particular values is extremely low. Therefore, the following equation holds:

$$\begin{aligned}\frac{\partial g}{\partial w_{ij}} &= -2w_{ij}\Theta(I_i)s_j^{pre}(t) + \\ &\left(1 - w_{ij}^2\right)s_j^{pre}(t)\frac{\partial\Theta(I_i)}{\partial I_i}\frac{\partial I_i}{\partial w_{ij}} \approx -2w_{ij}\Theta(I_i)s_j^{pre}(t).\end{aligned} \tag{4.11}$$

Using Eqs. 4.6, 4.8, and 4.11, we evaluate the updates on the weight $w_{ij}$ and the Lagrange multiplier $\lambda_{ij}$:

$$\begin{cases} \Delta w_{ij} = -\eta_w \dfrac{\partial L}{\partial w_{ij}} = -\eta_w\left(\dfrac{\partial l}{\partial w_{ij}} + \lambda_{ij}\dfrac{\partial g}{\partial w_{ij}}\right) \\ \qquad = -\eta_w\left(T_i(t) - 2\lambda_{ij}w_{ij}\right)\Theta(I_i)s_j^{pre}(t) \\ \Delta\lambda_{ij} = \eta_\lambda \dfrac{\partial L}{\partial \lambda_{ij}} = \eta_\lambda g = \eta_\lambda\left(1 - w_{ij}^2\right)\Theta(I_i)s_j^{pre}(t) \end{cases} \tag{4.12}$$

As highlighted in the previous section, the weights for the forward paths (to calculate current input $I_i$) and backward paths (to calculate $T_i$) are forcibly binarized (Eq. 4.7) to reduce the hardware computing workload.

The weights are initialized using the Xavier uniform initialization [21], whereas the Lagrange multipliers are initialized to zero. We confine each weight to between -1 and 1 by projecting *w* to -1 (1) when the updated weight is smaller than -1 (larger than 1). This weight clipping prevents unlimited weight growth. The eWB-eRBP algorithm is given in pseudocode in Algorithm 1.

**Algorithm 1** eWB-eRBP algorithm.

**Initialize** $\boldsymbol{w}$, $\boldsymbol{\lambda}$

**while** *True* **do**

$\boldsymbol{w}^b \leftarrow \text{Binarize}(\boldsymbol{w})$

**for** k $\in$ {presynaptic event indices $\boldsymbol{s}^{pre}$} **do**

**if** $b_{\min} < I < b_{\max}$ **then**

$\boldsymbol{w} \leftarrow \text{Clip}\{\boldsymbol{w} - \eta_w \nabla_{\boldsymbol{w}} L, -1, 1\}$

$\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \eta_\lambda \nabla_{\boldsymbol{\lambda}} L$

**end if**

**end for**

return $\boldsymbol{w}^{\mathbf{b}}$

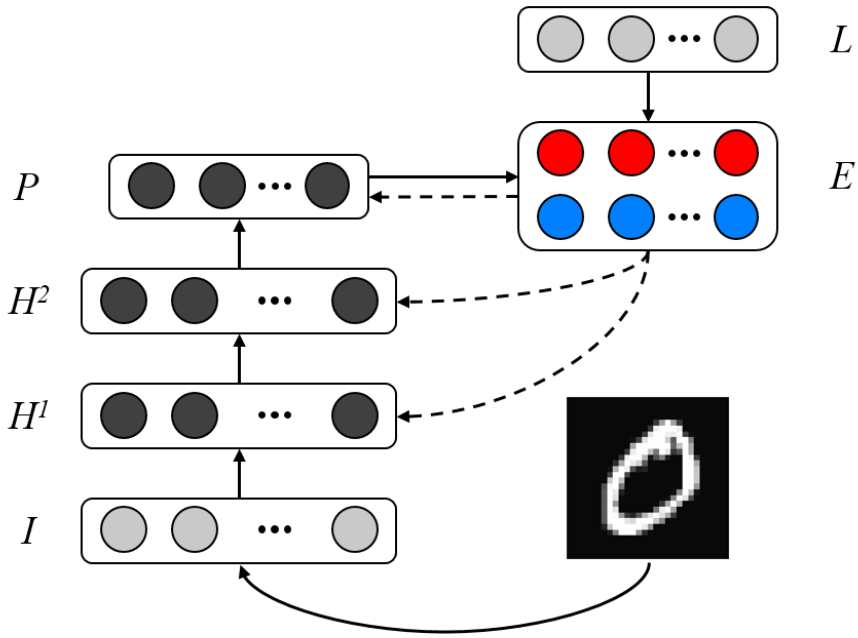


Figure 4.1. SNN architecture for eRBP. The error-coding layer (E) consists of two error-coding neurons for each label dimension that encode false positive and negative errors between labels (L) and predictions (P). During training, each of the hidden (in $H^1$ and $H^2$) and prediction (in P) neurons receives random feedback from the error neurons with fixed random weights (dashed arrows). The input layer is indicated by I.

### 4.2.4. Non-optimal weight binarization algorithm

The defining feature of eWB is the optimization of the binary weight distribution over the SNN. To highlight the performance of eWB, we compare eWB-eRBP with eRBP in conjunction with forced-to-be-binary weights conforming to Eq. 4.7, referred to as fWB-eRBP. Note that fWB stands for forced weight binarization. In fWB-eRBP, the binary weight distribution is non-optimal, and the real-valued weights are optimized using the loss function $l$ only. In eWB-eRBP, the real-valued weights are used for weight update only, and the signal forward propagation and error backpropagation use the binarized weights given by Eq. 4.7 instead. The fWB-eRBP algorithm is given in pseudocode in Algorithm 2.

**Algorithm 2** fWB-eRBP algorithm

**Initialize** $\boldsymbol{w}$

**while** *True* **do**

$\boldsymbol{w}^b \leftarrow \mathrm{Binarize}(\boldsymbol{w})$

**for** k $\in$ {presynaptic spike indices $\boldsymbol{s}^{pre}$} **do**

**if** $b_{\min} < I < b_{\max}$ **then**

$\boldsymbol{w} \leftarrow \mathrm{Clip}\{\boldsymbol{w} - \eta_1 \nabla_{\boldsymbol{w}} f, -1, 1\}$

**end if**

**end for**

return $\boldsymbol{w}^{\mathbf{b}}$

## 4.3. Results

We trained three types of fully connected SNNs (784-*h*-*h*-10; $h$ = 200, 500, and 1000) on MNIST. One training epoch consisted of 60,000 full training data that were selected randomly and input into the SNN. The intensity of each pixel in each hand-written digit image was encoded as the firing rate of input spikes (10–265 Hz) in proportion to the intensity. Note that even blank pixels were encoded at 10 Hz to serve as low-frequency background noise. Each image was shown to the SNN for 200 ms. To avoid interference from the previous training image, all neuronal

variables were reset to zero before the onset of the current training image. The classification accuracy was evaluated once every 500 training data using the 10,000 test data. The predicted output was identified by counting the number of spikes from each output neuron for 200 ms. The parameters used are listed in Table 4.1.

### 4.3.1. Classification accuracy

We used the three aforementioned algorithms (eRBP, eWB-eRBP, and fWB-eRBP) to train the three SNNs (784-*h*-*h*-10; $h = 200$, 500, and 1000). The final classification accuracy for each case was measured after the 25th training epoch (Table 4.2). For all three algorithms, the 784-1000-1000-10 SNN achieved the best accuracy. The accuracy evolution of this SNN for each algorithm is shown in Fig. 4.2. It is noted that weight binarization using either algorithm results in the loss of classification accuracy. Nevertheless, eWB-eRBP outperforms fWB-eRBP in terms of the loss for all SNNs. For example, for 784-1000-1000-10, the losses for eWB-eRBP and fWB-eRBP are 1.85% and 2.43%, respectively. This highlights the importance of optimal weight binarization for inference. Fig. 4.2 also shows that the fluctuations in accuracy over the inference period for eWB-eRBP are negligible compared with those for fWB-eRBP. This stability results from eWB asymptotically driving the real-valued weights toward the binary weights during training. Thus, the forced-to-be-binary weights conforming to Eq. 4.7 that are used for inference negligibly alter the accuracy over successive inference periods, particularly when the weights are close to binary values.

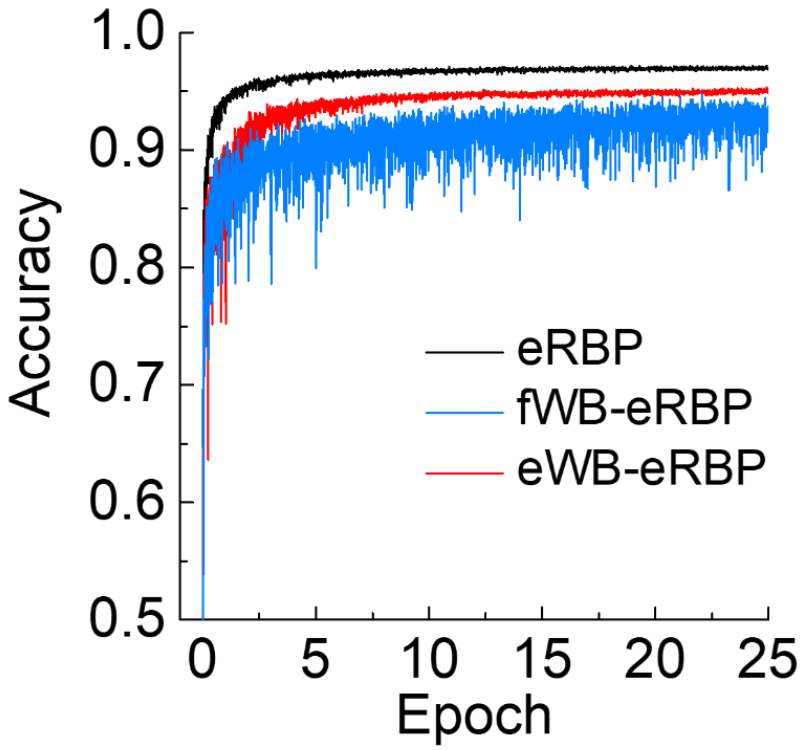


Figure 4.2. Classification accuracies of 784-1000-1000-10 SNNs on MNIST that were trained using eRBP, fWB-eRBP, and eWB-eRBP.

Table 4.1. Parameters for simulations.

| Symbol | Explanation | Value |
|---|---|---|
| $N_{\mathrm{d}}$ | Number of data neurons | 784 |
| $N_{\mathrm{h}}$ | Number of hidden neurons | 200,500,1000 |
| $N_{\mathrm{l}}$ | Number of label neurons | 10 |
| $N_{E+}$ | Number of positive error neurons | 10 |
| $N_{E-}$ | Number of negative error neurons | 10 |
| $N_p$ | Number of prediction neurons | 10 |
| $\tau_{refr}$ | Refractory period | 4 ms |
| $\tau_{syn}$ | Synaptic time constant | 4 ms |
| $g_V$ | Leak constant state $V$ | 1 nS |
| $g_{\mathrm{U}}$ | Leak constant state $U$ | 5 nS |
| C | Membrane capacitance | 1 pF |
| $V_{\mathrm{th}}$ | Threshold for spikes | 1.1 V |
| $w^E$ | Fixed weight | 1 nA |
| $b_{min}$, $b_{max}$ | Boxcar function constants | -25, 25 nA |
| $\eta_1$ | Learning rate | $2e^{-4}$ |
| $\eta_2$ | Lagrange multiplier step-size parameter | $2e^{-7}$ |

Table 4.2. Classification accuracy on the MNIST dataset for eRBP, eWB-eRBP, and fWB-eRBP after 25 epochs.

| | Classification accuracy (%) | | |
|---|---|---|---|
| Network | eRBP | eWB-eRBP | fWB-eRBP |
| 784-200-200-10 | 96.32 | 93.81 | 93.59 |
| 784-500-500-10 | 96.92 | 95.05 | 94.50 |
| 784-1000-1000-10 | 97.20 | 95.35 | 94.77 |

### 4.3.2. Weight binarization

To evaluate the degree of weight binarization during training, we introduce a constraint failure score (CFS) for a real-valued weight matrix $\boldsymbol{w} \in \mathbb{R}^{N\times M}$ as follows:

$$\mathrm{CFS} = 1 - \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M} w_{ij}^2$$

Therefore, when all weights are binarized, the CFS equals zero. We monitored the change in CFS over the training epoch (using eWB-eRBP or fWB-eRBP) for the 784-1000-1000-10 SNN. There are three weight matrices: $\boldsymbol{w}^{(hi)}$ (between the first hidden layer and the input layer), $\boldsymbol{w}^{(hh)}$ (between the second and first hidden layers), and $\boldsymbol{w}^{(oh)}$ (between the output layer and the second hidden layer). The changes in CFS for these matrices are shown in Figs. 4.3(a), (b), and (c), respectively. The CFS for eWB-eRBP asymptotically decreases to zero, ensuring successful weight binarization. For eWB-eRBP, the distributions of the trained real-valued weights in the weight matrices $\boldsymbol{w}^{(hi)}$, $\boldsymbol{w}^{(hh)}$, and $\boldsymbol{w}^{(oh)}$ are plotted in Figs. 4.3(d), (e), and (f), respectively. These distributions are compared with the distributions of the initial weights. Considering a weight $w$ ($|w| > 0.9$) to be fully binarized, the proportions of such fully binarized weights are 84.7%, 53.9%, and 72.9% in $\boldsymbol{w}^{(hi)}$, $\boldsymbol{w}^{(hh)}$, and $\boldsymbol{w}^{(oh)}$, respectively. The main cause of imperfect binarization is discussed in the following section. In contrast, the weight distribution for fWB-eRBP is rather diffusive over the entire weight range [Figs. 4.3(g)–(i)]. Consequently, the proportions of fully binarized weights after training are 30.0%, 27.1%, and 18.0% for $\boldsymbol{w}^{(hi)}$, $\boldsymbol{w}^{(hh)}$, and $\boldsymbol{w}^{(oh)}$, respectively.

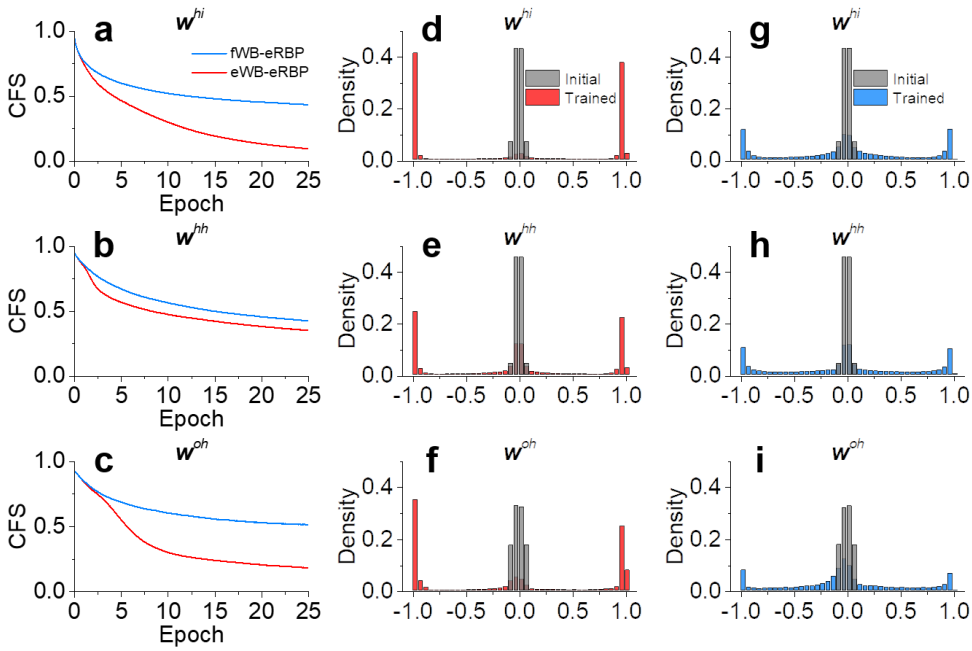


Figure 4.3. Weight distribution of eWB-eRBP and fWB-eRBP for $w^{(hi)}$ (between the first hidden layer and input layer), $w^{(hh)}$ (between the second and first hidden layers), and $w^{(oh)}$ (between the output layer and the second hidden layer). (a)–(c) Changes in CFS over epoch for eWB-eRBP and fWB-eRBP. The weight distribution of the initial and trained real-valued weights for (d)–(f) eWB-eRBP and (g)–(i) fWB-eRBP.

### 4.3.3. Computational complexity

Although the eWB algorithm is proposed for neuromorphic processors, for the moment, neuromorphic processors that serve as platforms for algorithm studies with high degrees of freedom are not available at hand. Instead, we used a GPU workstation (CPU: Intel Xeon Silver 4110 2.10GHz, GPU: RTX 2080 Ti). The algorithm was implemented in Python. Because eWB is not a standalone learning algorithm, we measured the time complexity of eWB from the difference in time complexity between RBP and eWB-eRBP. The eRBP and eWB-eRBP algorithms applied to a 784-500-500-10 SNN on MNIST for 25 learning epochs, yielding a wall-clock time of 4.99E5 s and 6.27E5 s, respectively (Table 4.3). The additional wall-clock to eRBP (1.28E5 s) arose from eWB. Additionally, we measured the space complexity for eRBP and eWB-eRBP, 366.0 and 368.0 MB, respectively. The MNIST dataset occupies 360.0 MB, so that eRBP and eWB-eRBP occupy 6.0 and 8.0 MB, respectively.

Akin to multiply-accumulate operations (MACs) for deep learning implemented in general-purpose hardware, synaptic operations (SynOps) in neuromorphic hardware are known to consume considerable power, so that the number of SynOps can be a relative measure of energy-efficiency for learning. We evaluated the number of SynOps required for training a 784-500-500-10 SNN on MNIST using eWB-eRBP. The SNN was trained for 25 epochs in aggregate. Fig. 4.4 shows the evaluated number of SynOps and classification accuracy for each learning epoch.

For a comparison with binarized neural network (BNN) [18], we measured the number of MACs required for training a 784-500-500-10 BNN on MNIST. Each MNIST image was pre-binarized to ±1 using the sign function. We used Batch Normalization with a minibatch size of 100. The square hinge loss was minimized using Adam optimizer. We employed an exponentially decaying global learning rate and Glorot initialization. Dropout layers were deployed to regularize the BNN. The evaluation results are co-plotted in Fig. 4.4, indicating that both networks require similar same numbers of operations to reach an accuracy approximately 0.93.

The comparison indicates similar operational complexity for both cases. Yet, power-efficiency for eWB-eRBP likely outperforms BNN when implemented in neuromorphic hardware. Power-efficiency is the defining feature of neuromorphic hardware. For instance, Loihi (digital neuromorphic processor) [22] highlights high power-efficiency, approximately 300 times that of graphics processing units [23]. Thus, we expect a two orders of magnitude increase in power-efficiency when eWB-eRBP is embedded in neuromorphic hardware.

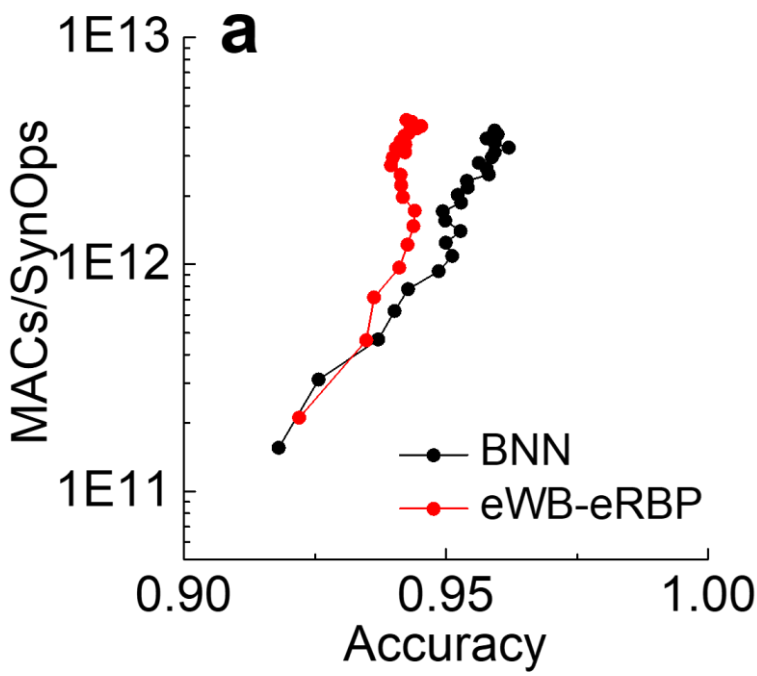


Figure 4.4. Efficiency in learning. Number of SynOps for a 784-500-500-10 SNN with eWB-eRBP algorithm. BNN (784-500-500-10 network) is compared with the eWB-eRBP in terms of the number MACs required for reaching a given accuracy for the MNIST learning task.

Table 4.3. Time and space complexity on the MNIST dataset for 784-500-500-10 eRBP and eWB-eRBP after 25 epochs.

| | eRBP | eWB-eRBP |
|---|---|---|
| Accuracy | 96.92 | 95.05 |
| Wall-clock time (s) | 4.99E5 | 6.27E5 |
| Memory usage (MB) | 366.0 | 368.0 |

## 4.4. Discussion

Generally, event-based learning algorithms update a weight only if a presynaptic or postsynaptic event (local to the synapse) occurs, unlike BP, which updates all weights layer-wise. Specifically, eWB-eRBP addresses only the synapses that satisfy the two conditions of (i) presence of presynaptic spike, and (ii) non-zero boxcar function of the postsynaptic activation, as described in Eq. (12). During training, several synapses were inactive (and their presynaptic neurons quiescent), and thus, they maintained their initial weights until the end of training. The high proportion of fully binarized weights in w(hi) (84.7%) is due to the blank pixels being encoded at a 10-Hz spike rate rather than being left inactive.

Table 4.4 presents a comparison of the performance of eWB-eRBP with that of relevant works using limited-precision weights ($\leq$ 8-bit). For a fair comparison, we chose event-based algorithms applied to fully connected SNNs. Notably, most of them use higher precision than 1-bit. Nevertheless, the classification accuracy is lower than or only slightly better than that of our work. This highlights the performance of eWB.

The works by Yousefzadeh *et al.* [7] and Srinivasan and Roy [8] partly use 1-bit weights, but the usage is limited to only the weights between the input and hidden layers. The weights between the hidden and output layers are of higher precision to minimize the classification accuracy loss. Therefore, eWB is the first event-driven weight binarization algorithm with locality that ensures high performance.

The eWB-eRBP algorithm is an example to demonstrate the compatibility of eWB with event-based learning algorithms. In principle, eWB can also be combined with other event-based learning algorithms with appropriate modifications and can serve as a common weight binarization algorithm for various event-based learning algorithms. In this regard, attention should be paid to the performance reduction resulting from optimal weight binarization instead of the absolute performance when evaluating the performance of eWB. This is because the absolute performance is mainly determined by the learning algorithm combined with eWB.

Although we have applied LMM to weight binarization in this study, any other constraints can be considered as long as they are mathematically well-defined. For instance, ternary weight (0, ±1; 2-bit precision) constraints with eRBP can be formulated as $g(w_{ij}) = w_{ij}(1 - w_{ij}^2)\Theta(I_i)s_j^{pre}(t)$ instead of Eq. 4.10. This constraint function outputs zero when $w_{ij} = 0$ or $w_{ij} = \pm 1$, enabling the algorithm to learn optimal ternary weights. Further, 3-bit weight (0, ±1, ±2, ±3) constraints can be considered using the constraint function $g(w_{ij}) = w_{ij}(1 - w_{ij}^2)(4 - w_{ij}^2)(9 - w_{ij}^2)\Theta(I_i)s_j^{pre}(t)$. Therefore, the proposed LMM-based learning algorithm forms the foundation for event-based learning with various constraints.

The use of limited-precision weights improves not only memory efficiency but also energy efficiency. Energy efficiency is a key attribute of neuromorphic computing and is a defining motivation for event-based learning algorithms as alternatives to layer-wise synchronous learning such as BP. In digital neuromorphic hardware, a lower precision of the data format reduces the energy consumed in arithmetic operations. Horowitz [24] identified a 30-fold (18.5-fold) improvement in the energy efficiency by replacing 32-bit floating-point data with 8-bit fixed-point data in addition (multiplication) operations. The use of binary weights completely avoids the multiplication of weights and low-pass filtered spikes, which are otherwise needed for every synaptic operation. Given that synaptic operations impose the most significant workload on neuromorphic hardware, as is the case for multiply-accumulate operations in DNNs [16], SNNs with binary weights can achieve a large improvement in energy efficiency. Nevertheless, the degree of improvement depends on neuromorphic hardware design, which is not specified in this study.

Table 4.4. Comparison of reported classification accuracy of quantized fully connected SNNs on the MNIST dataset.

| Learning algorithm | Structure | Weight precision | Accuracy (%) |
|---|---|---|---|
| CD + BP [25] | 784-500-500-10 | 4-bit | 91.35 |
| STDP + eCD [16] | 784-500-10 | 4-bit | 94.80 |
| eRBP [9] | 784-200-200-10 | 8-bit | 96.50 |
| STDP + BP [7] | 784-6400-10 | 1-bit + 24-bit | 95.70 |
| STDP + BP [8] | 784-6400-10 | 1-bit + 32-bit | 92.14 |
| eWB-eRBP (This work) | 784-1000-1000-10 | 1-bit | 95.35 |

## 4.5. Conclusion

In this study, we proposed an eWB algorithm that optimally binarizes weights in an SNN based on local events. The optimal configuration of binary weights is calculated using the LMM with binary weight constraints. Given that eWB addresses local data only to update weights in an event-based manner, it is inherently compatible with multicore neuromorphic hardware. When combined with an event-based learning algorithm using an appropriate loss function, eWB enables the network to learn binary weights that minimize the loss function. This was demonstrated using eWB-eRBP (eWB combined with eRBP), which was applied to train fully connected SNNs on MNIST. The consequent classification accuracy is 95.35%, whereas eRBP with 32-bit weights yielded an accuracy of 97.20%. The results indicate an accuracy reduction of 1.85% as the cost of optimal weight binarization. To the best of our knowledge, eWB is the first method to learn binary weights based on events; therefore, a comparison with directly related methods is unavailable at the moment. Nevertheless, to highlight the importance of optimal binary weights in performance, eWB-eRBP was compared with fWB-eRBP (with non-optimal binary weights that were forcibly binarized) and was shown to yield better performance and more stable performance evolution over the training epoch than fWB-eRBP.

Finally, eWB is scalable to any event-based learning algorithm with appropriate modifications, thus serving as a common weight binarization method. The LMM is also scalable to any weight constraint as long as the constraint functions are mathematically well-defined. The eWB algorithm is an example that demonstrates this scalability.

## 4.6. Appendix

The eRBP algorithm is a presynaptic event-driven local learning rule that uses direct feedback alignment (Fig. 4.1). In eRBP, the weight update with a mean-squared loss function is formulated as

$$\Delta w_{ij}(t) = -T_i(t)\Theta(I_i)s_j^{pre}(t), \tag{4.13}$$

which realizes a three-factor rule with (i) presynaptic spike ($s_j^{\text{pre}}$), (ii) postsynaptic signal Θ, corresponding to the derivative of the postsynaptic activation, and (iii) error signal $T_i$, which backpropagates through random feedback channels.

(i) The presynaptic spikes are the output of neuron *i,* which is modeled using an LIF model that includes two defining variables, namely, the synaptic current $I_i$ and subthreshold somatic membrane potential $V_i$:

$$\begin{cases} \tau_{syn}\dfrac{d}{dt}I_i = -I_i + \sum_j w_{ij}s_j(t)\xi(t) \\ \mathrm{C}\dfrac{d}{dt}V_i = -g_V V_i + I_i \end{cases}$$

where $w_{ij}$, $s_j$, and $\xi$ denote the weight between neurons $j$ and $i$, spikes from neuron $j$, and a stochastic Bernoulli process with probability $(1 - p)$, respectively. The time constant for the synaptic current is denoted by $\tau_{\text{syn}}$. The ion conductance through the membrane is denoted as $g_V$.

(ii) As a workaround for the postsynaptic activation being non-differentiable, the derivative of the postsynaptic activation is approximated as a boxcar function Θ:

$$\Theta(I_i) = \begin{cases} 1 \text{ if } b_{min} < I_i < b_{max} \\ 0 \text{ otherwise} \end{cases}$$

This corresponds to the derivative of a hard sigmoid function with two transition points ($b_{\text{min}}$ and $b_{\text{max}}$).

(iii) The error signal $T_i$ is formulated as

$$T_i(t) = \textstyle\sum_k e_k(t)g_{ik}, \tag{4.14}$$

where $e_k$ is the error signal from the error-coding neuron $k$. The constant $g_{ik}$ denotes the fixed random feedback weight from the error-coding neuron $k$ to the hidden neuron $i$. It is noteworthy that this error signal is non-local to the synapse $w_{ij}$, and thus unavailable for updating the weight $w_{ij}$ using Eq. (13). It is conceivable that the

data may be moved from the location of error evaluation to the synapse during updating; however, this is not an optimal strategy for neuromorphic hardware in which neurons communicate using events only. To render the error local to the target synapses, eRBP uses two error-coding neurons with somatic potentials $V^{E+}$ and $V^{E-}$ for each output dimension. They code for false positive and negative errors, respectively. Their subthreshold behaviors are modeled using a perfect integrate-and-fire model

$$\mathrm{C}\frac{d}{dt}V^{\mathrm{E}\pm} = \pm w^{E}(s^{P}(t) - s^{L}(t)),$$

where $s^P$ and $s^L$ are the spike trains from the prediction neurons and labels, and $w^E$ is a positive constant. The false positive error coding neuron (potential $V^{E+}$ and weight $w^E$) spikes and generates the spike train $s^{E+}$ when $s^p = 1$ and $s^L = 0$, whereas the false negative error coding neuron (potential $V^{E-}$ and weight $-w^E$) spikes and generates the spike train $s^{E-}$ when $s^p = 0$ and $s^L = 1$. The consequent spike trains $s_j^{E+}$ and $s_j^{E-}$ (from the two error-coding neurons for label $j$), rather than the error data themselves, are relayed to the target synapses through the random weight $g_{ij}$ so that the communication architecture is well suited for neuromorphic hardware.

The error spike trains $s_j^{E+}$ and $s_j^{E-}$ from label $j$ are subsequently encoded as firing rates to eventually realize the error signal $T_i$ in Eq. (14). To this end, each neuron in the output and hidden layers is given a dendritic compartment that calculates the dendritic potential ($U_i^h$ for hidden neuron $i$ and $U_i^p$ for prediction neuron $i$) using a leaky integrated model

$$C\frac{d}{dt}U_i^h = -g_U U_i^h + \sum_j g_{ij}(s_j^{E+}(t) - s_j^{E-}(t))$$

and

$$C\frac{d}{dt}U_i^p = -g_U U_i^p + w^E(s_i^{E+}(t) - s_i^{E-}(t))$$

This dendritic potential is equivalent to the error signal $T_i$ and is local to each target synapse. Therefore, the learning rule in Eq. (13) can be rewritten as

$$\Delta w_{ij} = \eta U_i \Theta(I_i) s_j(t)$$

The parameters used in this study are listed in Table 4.1.

## 4.7. Bibliography


[1] M. Pfeiffer and T. Pfeil, *Frontiers in neuroscience,* vol. 12, p. 774, 2018.

[2] D. S. Jeong, *Journal of Applied Physics,* vol. 124, no. 15, p. 152002, 2018.

[3] K. Roy, A. Jaiswal, and P. Panda, *Nature,* vol. 575, no. 7784, pp. 607-617, 2019.

[4] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, *Frontiers in neuroscience,* vol. 13, p. 95, 2019.

[5] C. Lee, S. S. Sarwar, and K. Roy, 2019, *arXiv:1903.06379.*

[6] G.-q. Bi and M.-m. Poo, *Journal of neuroscience,* vol. 18, no. 24, pp. 10464-10472, 1998.

[7] A. Yousefzadeh, E. Stromatias, M. Soto, T. Serrano-Gotarredona, and B. Linares-Barranco, *Frontiers in neuroscience,* vol. 12, p. 665, 2018.

[8] G. Srinivasan and K. Roy, *Frontiers in Neuroscience,* vol. 13, p. 189, 2019.

[9] E. O. Neftci, C. Augustine, S. Paul, and G. Detorakis, *Frontiers in neuroscience,* vol. 11, p. 324, 2017.

[10] D. Kim, V. Kornijcuk, C. S. Hwang, and D. S. Jeong, *IEEE Access,* vol. 8, pp. 110523-110534, 2020.

[11] F. Ponulak and A. Kasiński, *Neural computation,* vol. 22, no. 2, pp. 467-510, 2010.

[12] R. Gütig and H. Sompolinsky, *Nature neuroscience,* vol. 9, no. 3, pp. 420-428, 2006.

[13] S. M. Bohte, J. N. Kok, and H. La Poutre, *Neurocomputing,* vol. 48, no. 1-4, pp. 17-37, 2002.

[14] M. Suri *et al.*, *IEEE Transactions on Electron Devices,* vol. 60, no. 7, pp. 2402-2409, 2013.

[15] E. Neftci, S. Das, B. Pedroni, K. Kreutz-Delgado, and G. Cauwenberghs, *Frontiers in neuroscience,* vol. 7, p. 272, 2014.

[16] E. O. Neftci, B. U. Pedroni, S. Joshi, M. Al-Shedivat, and G. Cauwenberghs, *Frontiers in neuroscience,* vol. 10, p. 241, 2016.

[17] W. Gerstner and W. M. Kistler. Cambridge university press, 2002.

[18] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, 2016, *arXiv:1602.02830*.

[19] M. Maher and J.-F. Puget. Springer Science & Business Media, 1998.

[20] J. C. Platt and A. H. Barr, *Proceedings of the 1987 International Conference on Neural Information Processing Systems*, 1987.

[21] X. Glorot and Y. Bengio, *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249-256.

[22] M. Davies *et al.*, *IEEE Micro,* vol. 38, no. 1, pp. 82-99, 2018.

[23] P. Blouw, X. Choo, E. Hunsberger, and C. Eliasmith, *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop*, 2019, pp. 1-8.

[24] M. Horowitz, in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014: IEEE, pp. 10-14.

[25] E. Stromatias, D. Neil, M. Pfeiffer, F. Galluppi, S. B. Furber, and S.-C. Liu, *Frontiers in neuroscience,* vol. 9, p. 222, 2015.

# 5. Conclusion

Neuromorphic hardware capable of parallel computation enables the efficient computation of neuronal variables in SNN. With event-driven weight updates using local data leverage the capacity of neuromorphic hardware. Parallel computation can be accelerated when neuromorphic hardware is combined with a binary resistance switch array to implement an artificial synaptic array. Thus, we have studied on two subjects, which are a sequence-predicting SNN architecture with event-based learning algorithm and method for training SNN with binary weight in an event-driven fashion.

At first, we introduce an SNN architecture for sequence predictions (*n*-SPSNN) by deploying working memories and a novel learning algorithm (LbAP algorithm) for SNNs, which is suitable for neuromorphic hardware. We demonstrate the sequence-learning capability of the proposed architecture and learning algorithm using the Nottingham dataset and random sequences. The *n*-SPSNN shows high tolerance to errors in input encoding, which is higher than the state-of-the-art sequence learning hypotheses LSTM and GRU. Also, *n*-SPSNN offers high tolerance to random changes in input-encoding delay. The LbAP algorithm endows the *n*-SPSNN with efficiency in learning. The learning is completed in a few iterations. Therefore, the LbAP algorithm can train SNNs in an energy- and time-efficient manner.

Secondly, we introduce a novel weight binarization algorithm (eWB), which is an event-driven algorithm with the locality. The binary weights can not only be applied to binary resistance switch arrays in digital neuromorphic hardware but also reduce energy consumption. LMM with binary weight constraints is used to optimize the weights to satisfy the constraint condition. We elaborate on the scaling of eWB to eRBP (eWB-eRBP), which was applied to train fully connected SNNs on MNIST. The classification accuracy of the 784-1000-1000-10 network is 95.35% and shows an accuracy reduction of 1.85% as the cost of optimal weight binarization. Optimal binarization endows eWB with better and more stable performance evolution over

the training age when compared to fWB-eRBP (with coercively binarized non-optimal binary weights).

# Abstract (in Korean)

스파이킹 신경망은 생물학적 관찰에 기반한 신경망 모델로 에너지 효율적인 계산이 가능한 것으로 알려져 있다. 스파이킹 신경망에서 뉴런의 내부 상태는 다른 뉴런에서 오는 스파이크와 시간 정보에 따라 변하며 뉴런의 막전위가 특정 임계값을 초과할 경우 스파이크가 발생한다. 뉴런의 스파이크는 시공간 상에서 드물게 발생하며 이벤트에 기반해 정보를 전달한다. 정보 밀도가 높은 스파이크에 기반한 스파이킹 신경망은 순열과 같이 시공간 상에 분포된 스파이크 패턴의 효과적인 학습을 가능하게 한다. 그러나 스파이킹 신경망 계산을 위해서는 매 단위 시간마다 막 전위와 같은 내부 상태 업데이트가 필요하며 이는 많은 계산 시간을 요구한다. 이러한 이유로 효율적인 스파이킹 시뮬레이션을 하기 위해서는 뉴런의 내부 상태 업데이트가 병렬적으로 처리되어야 한다. 분산 프로세서와 로컬 메모리 구조를 가지는 뉴로모픽 하드웨어는 병렬 계산을 가능하게 하며 이벤트 기반 지역 정보를 통해 스파이킹 신경망을 학습시킬 경우 그 효율성이 극대화된다. 그러나 범용적인 이벤트와 지역 정보에 기반한 스파이킹 신경망 학습 방법은 아직 존재하지 않으며 특히, 연관 리콜에 관한 연구는 아직 이뤄지지 않고 있다.

이 논문에서는 단일 단계 예측 및 시퀀스 간 예측, 즉 연관 리콜이 가능한 *n*-SPSNN (*n*th order-predicting SNN)을 소개한다. 이러한 기능의 핵심으로, LbAP (learning by backpropagating action potential) 알고리즘이라는

새로운 학습 알고리즘을 제안한다. LbAP 알고리즘은 (i) 시냅스 후 이벤트 기반 학습 (ii) 시간적 로컬 데이터만 사용 iii) 경쟁으로 인한 가중치 정규화 효과 (iv) 빠른 학습의 특징을 가지고 있다. 가장 중요한 것은 LbAP 알고리즘이 로컬 데이터만을 이용해 전체 SPSNN에 대한 통합 학습 프레임워크를 제공한다는 것이다. SPSNN의 학습 능력은 주로 은닉층 뉴런의 수 $h$에 의해 결정된다. 은닉 뉴런 수 $h$가 학습 시퀀스 길이 $l$의 두 배보다 클 때 시퀀스 예측 정확도는 최대 값 (~ 1)에 도달한다. 또한 SPSNN은 최신의 시퀀스 학습 네트워크인 LSTM (long short-term memory) 및 GRU (gated recurrent unit)에 비해 입력 인코딩 오류에 대한 높은 내성을 가진다. 성공적인 학습을 위해 필요한 SPSNN 시냅스 동작 수와 LSTM 및 GRU의 행렬 곱 수를 비교한 결과, SPSNN은 다른 두 네트워크에 비해 약 100배의 효율성을 보였다. SPSNN의 높은 효율성은 LbAP 알고리즘에 기인하는 SPSNN의 빠른 학습에서 비롯된 것으로 볼 수 있다.

뉴로모픽 하드웨어에 저항 스위치와 같은 비휘발성 메모리를 적용할 경우 더욱 효율적인 신경망 학습을 구현할 수 있다. 저항 스위치 어레이의 행렬-벡터 곱셈은 신경망에서의 스파이크 전파 과정과 유사하다. 입력 전압 벡터에 대한 전류 응답을 측정함으로써 뉴로모픽 하드웨어에서의 행렬-벡터 곱셈을 효율적으로 처리할 수 있다. 따라서 이진 저항 스위치 어레이로 인공 시냅스 어레이를 구현할 경우, 기존

네트워크에서의 행렬-벡터 계산을 병렬적으로 처리해 계산 속도를 높일 수 있다.

그러나 기존의 스파이킹 신경망은 시냅스 가중치를 저장하기 위해 다중 비트가 필요하다. 이는 이진 저항 어레이를 이용한 뉴로모픽 하드웨어의 시냅스 어레이 구현을 어렵게 만든다. 또한 다중 비트 정밀도를 사용하면 메모리 사용량이 증가하고 복잡한 패턴 인식 작업 시 스파이킹 신경망의 계산 효율성을 감소시킨다.

따라서 본 연구에서는 이진 시냅스 가중치 (-1, 1)를 가진 스파이킹 신경망에 대한 새로운 이벤트 기반 가중치 이진화 알고리즘을 제안한다. eWB (event-based weight binarization algorithm for spiking neural networks) 알고리즘은 주어진 제약 내에서 매개 변수를 최적화하는 라그랑주 승수법을 기반으로 한다. eWB 알고리즘은 (i) 지역 정보만을 통해 이벤트 기반 점차적 가중치 이진화 (ii) eRBP (event-driven random backpropagation)와 같은 이벤트 기반 학습 알고리즘과 완전한 호환성 (iii) 이진 가중치 제한 조건을 포함한 다양한 가중치 제한 조건을 처리할 수 있다. 이를 증명하기 위해 eWB와 eRBP를 결합한 이진 가중치를 학습하는 단일 알고리즘인 eWB-eRBP 구현했다. 완전 연결을 가지는 스파이킹 신경망에서 eWB-eRBP를 MNIST를 학습시켰을 시 95.35%의 정확도를 달성했다.

**주요어:** 뉴로모픽 엔지니어링, 저항 변화 스위치, 스파이킹 신경망, 이벤트 기반 지역성 학습 알고리즘, 순차 학습, 연관 리콜, 이벤트 기반 가중치 이진화 학습 알고리즘

**학번:** 2016-20771

김 도 헌